Departamento de Sistemas y Automática
Ingeniería en Tecnologías Industriales
Universidad Carlos III de Madrid

Generación de trayectorias mediante MoveIt para ASIBOT 2

Trabajo de Fin de Grado

Autora: Celia Nieto Agraz
Tutor: Alberto Jardón Huete
Fecha de entrega: 26 Septiembre, 2017

Agradecimientos

A mis padres, por todo su esfuerzo, sacrificios, paciencia, amor y apoyo.

A mi hermana y Pablo, por todo su cariño y apoyo.

A Alberto, por darme la oportunidad de introducirme en el mundo de la robótica asistencial.

A Jorge y Bartosz, por su paciencia y ayuda para solucionar mis dudas.

A todos mis amigos del Grado, sin los cuales estos cuatro años habrían sido mucho más duros y menos interesantes.

Resumen

El proyecto consiste en la adaptación de entornos de simulación para el robot ASIBOT, desarrollados para la herramienta de simulación robótica OpenRave, con el objetivo de poder trabajar con estos mismos en el entorno de Gazebo.

En la primera parte del documento se analizan y comparan ambos modelos de simulación y se explica el proceso seguido para la transformación de los archivos.
Como resultado de la adaptación se consiguen entornos y elementos robóticos funcionales en Gazebo con características similares a los originales tomados de OpenRave.

La segunda parte del documento se centra en la creación de trayectorias asistenciales para realizar con el robot ASIBOT por medio de la herramienta MoveIt, que interactúa con el entorno diseñado en Gazebo.

Abstract

The aim of this project is to convert the models of simulation for the ASIBOT robot created for the simulation tool OpenRave in order to work with them in Gazebo.

The first part of this document analyzes and compares both simulation models and explains the process followed to transform the files.
As a result of this adaptation, robotics environments and elements that are functional in Gazebo are obtained, with similar characteristics to the originals from OpenRave.

The second part of the document focuses on the creation of welfare trajectories for the ASIBOT robot using the tool MoveIt, that interacts with the environment designed in Gazebo.

Índice general

Agradecimientos **II**

Resumen **III**

Abstract **IV**

Lista de Figuras VIII

Lista de Tablas XI

1. Introducción **1**

1.1. Robótica asistencial en la Universidad Carlos III 2

1.1.1. Robot humanoide NAO 2

1.1.2. Maggie 3

1.1.3. Brazo asistencial AMOR 4

1.1.4. Brazo asistencial ASIBOT 4

1.2. Simulación 9

1.3. Motivación del proyecto 10

1.4. Objetivo del proyecto 11

1.5. Estructura 11

2. Estado del arte **12**

2.1. ROS: *Robot Operating System* 13

2.1.1. Nivel sistema de archivos (*Filesystem level*) 14

2.1.2. Nivel computación gráfica (*Computation graph*) 15

2.1.3. Nivel comunitario *(Community level)* 16

2.2. Simuladores robóticos: OpenRave y Gazebo 17

2.2.1. OpenRave 17

2.2.2. Gazebo 17

2.3. Creación de entornos: Comparación entre Gazebo y OpenRave 18

2.3.1. Definición del entorno 18

2.3.2. Formato de lenguaje 19

2.3.3. Definición de archivos 22

2.3.4. Elementos del entorno 27

3. Arte del modelado **36**
3.1. Archivos originales . . . 37
3.2. Archivos finales . . . 37
3.3. Transformación de .xml a .sdf . . . 38
3.3.1. Encabezado del archivo . . . 38
3.3.2. Eslabones . . . 39
3.3.3. Articulaciones . . . 40
3.3.4. Transformación de mallas de .iv a .stl . . . 41
3.3.5. Archivos del entorno . . . 45
3.3.6. Resultados . . . 46

4. Creación de Trayectorias **48**
4.1. Transformación de archivos .sdf a .urdf . . . 49
4.1.1. Paquete *pysdf* . . . 49
4.1.2. Conversión de los archivos . . . 50
4.2. Asistente de MoveIt y planificación de trayectorias con Rviz . . . 51
4.2.1. MoveIt . . . 51
4.2.2. Rviz . . . 57
4.3. Conexión Gazebo y Rviz . . . 64
4.3.1. Creación de archivos . . . 64

5. Resultados y conclusiones **70**
5.1. Resultados del trabajo . . . 71
5.1.1. Trayectoria rehabilitación . . . 71
5.1.2. Trayectoria en la cocina . . . 73
5.1.3. Trayectoria en el baño . . . 75
5.2. Análisis crítico de los resultados . . . 77
5.3. Conclusiones . . . 78

6. Mejoras y trabajos futuros **79**

Apéndices **81**

A. Marco Regulador **82**

B. Entorno Socio-Económico **83**
B.1. Presupuesto . . . 83

C. Códigos **84**
C.1. Código ASIBOT definido en OpenRave . . . 84
C.2. Código ASIBOT definido en Gazebo . . . 86
C.3. Código cocina definido en OpenRave . . . 97
C.4. Código cocina definido en Gazebo . . . 97
C.5. Código ASIBOT garra definido en URDF . . . 98

Bibliografia **102**

Índice de figuras

1.1. Sesión de rehabilitación con NAO 3
1.2. Robot Maggie 3
1.3. Robot AMOR 4
1.4. Robot ASIBOT 5
1.5. Movimiento entre los anclajes mecánicos 7
1.6. Herramientas ASIBOT 7
1.7. Cocina del laboratorio Robotics Lab 8

2.1. Jerarquía del sistema de archivos 14
2.2. Nivel de computación gráfica 16
2.3. Entorno en Gazebo 19
2.4. Entorno en OpenRave 19
2.5. Etiqueta que define el encabezado y versión de un archivo .sdf 21
2.6. Etiqueta que define el archivo de entorno en OpenRave 22
2.7. Etiqueta para definir el archivo de entorno en Gazebo 22
2.8. Etiqueta para definir un modelo en OpenRave 23
2.9. Etiqueta para incluir un modelo en el archivo de entorno en OpenRave . 23
2.10. Etiqueta para definir un modelo en Gazebo 24
2.11. Etiqueta para incluir un modelo en el archivo de entorno en Gazebo . . . 24
2.12. Etiqueta para definir un robot en OpenRave 24
2.13. Etiqueta para incluir un robot en el archivo de entorno en OpenRave . . 24
2.14. Etiqueta para definir un modelo dentro de un archivo .urdf 25
2.15. Etiqueta para incluir una malla en OpenRave 26
2.16. Etiqueta para incluir una malla en Gazebo 26
2.17. Comparación de código de escalas 27
2.18. Comparación de código de cuerpos o *links* 28
2.19. Comparación de código de geometría visual y de colisión 29
2.20. Comparación de código de los tipos de geometría 30
2.21. Comparación de código de traslación y rotación 31
2.22. Comparación de código de propiedades dinámicas 32
2.23. Comparación de código de articulaciones o *joints* 34

3.1. Esquema ASIBOT 38
3.2. Encabezado del archivo .xml y .sdf 39
3.3. Comparación de la definición de un eslabón en el archivo .xml y .sdf . . . 39

3.4. Comparación de la definición de una articulación en el archivo .xml y .sdf 41
3.5. Importación del archivo .iv 44
3.6. Carga del fichero .iv 44
3.7. Exportación al formato .stl 44
3.8. Carga del archivo .stl de *Polytrans* 44
3.9. Relleno de los huecos 44
3.10. Exportación a formato .stl binario 44
3.11. Comparación entre los archivos de *Polytrans* y *Freeform* 45
3.12. Modelo en 3D original del baño y una vez realizado el corte 46
3.13. ASIBOT en el entorno de la cocina completo 46
3.14. ASIBOT en el entorno de la cocina, vista lateral 46
3.15. ASIBOT en el entorno del baño 47

4.1. Error al cargar el fichero 50
4.2. Definición de la dirección al fichero 51
4.3. Lanzamiento del asistente de MoveIt y carga del archivo 52
4.4. Generación de la matriz de colisiones 53
4.5. Definición de articulaciones virtuales 53
4.6. Parámetros del grupo 54
4.7. Elección de articulación del grupo pinza 54
4.8. Cadena cinemática para el grupo *arm* 54
4.9. Lista de los grupos 55
4.10. Definición de una posición para el robot 55
4.11. Definición de herramientas extremas 56
4.12. Generación de los archivos de configuración 56
4.13. Entorno de Rviz para generar trayectorias 58
4.14. Error de conexión al lanzar el archivo 59
4.15. Advertencia vinculada a las articulaciones virtuales 59
4.16. Añadir modelos al entorno en Rviz 61
4.17. Ejemplo de posición inicial y final para ASIBOT 63
4.18. Colisión en el entorno del baño 63
4.19. Colisión en el entorno de la cocina 63
4.20. Advertencia sobre el *controller_manager* 66
4.21. Código añadido al .urdf para cargar el plugin gazebo_ros_control 66
4.22. Error controladores 67
4.23. Fragmento del código asociado al *hardware* añadido al .urdf 67
4.24. Salida por terminal sin avisos 68
4.25. Servicios y controladores activos durante la conexión 68

5.1. Posición inicial y final subtrayectoria 1 de rehabilitación 72
5.2. Recorrido realizado por ASIBOT en la subtrayectoria 1 de rehabilitación 72
5.3. Posición inicial y final subtrayectoria 2 de rehabilitación 73
5.4. Recorrido realizado por ASIBOT en la subtrayectoria 2 de rehabilitación 73
5.5. Posición inicial y final subtrayectoria 1 de la cocina 74

5.6. Recorrido realizado por ASIBOT en la subtrayectoria 1 de cocina 74
5.7. Posición inicial y final subtrayectoria 2 de cocina 75
5.8. Recorrido realizado por ASIBOT en la subtrayectoria 2 de cocina 75
5.9. Posición inicial y final subtrayectoria 1 del baño 75
5.10. Recorrido realizado por ASIBOT en la subtrayectoria 1 del baño 76
5.11. Posición inicial y final subtrayectoria 2 del baño 76
5.12. Recorrido realizado por ASIBOT en la subtrayectoria 2 del baño 77

Índice de tablas

2.1. Correspondencias entre etiquetas y atributos en articulaciones 35

3.1. Diferencias en la definición de un elemento en .xml y .sdf 40
3.2. Diferencias en la definición de una articulación en .xml y .sdf 41

4.1. Jerarquía de los archivos necesarios para la conexión Rviz-Gazebo 65

B.1. Presupuesto del proyecto 83

Capítulo 1

Introducción

El *Robot Institute of America* definió en 1979 un robot como: "*A reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks*", lo cual puede ser traducido como: "Un operador multifuncional y reprogramable diseñado para mover materiales, piezas, herramientas o recursos especializados por medio de varios movimientos programados para el desarrollo de diferentes tareas".

Desde el siglo XIII se tiene constancia de autómatas que trataban de imitar el comportamiento humano, entre ellos se encuentra el Escriba automático diseñado por Pierre y Henri-Louis Jacquet-Droz en el siglo XVIII, las muñecas parlantes diseñadas por Edison en 1890 o el teleautomaton (primer dron marino) desarrollado por Tesla en 1898.
Sin embargo, la primera aparición de la palabra robot se dio en la obra de teatro R.U.R (*Robots Universalis Rosum*) del autor checo Karel Capek. La palabra original "*robota*" tenía un significado de servidumbre y esclavitud, este concepto dio lugar a los primeros intentos de crear máquinas autónomas con forma humana que obedeciesen a las órdenes de las personas.

A partir de este momento se comienza un desarrollo más intenso de esta ciencia, que a día de hoy evoluciona de forma constante por medio de grandes avances en la investigación, siendo cada vez mayor la integración de los robots dentro de la vida de las personas, tanto dentro de la industria, para conseguir un trabajo más seguro y eficiente o para operar dentro de ambientes que son extremadamente peligrosos para los humanos; también en el ámbito doméstico, ya sea por medio de aparatos robóticos orientados a las tareas del hogar como pueden ser lavadoras, lavavajillas o robots de cocina o aquellos orientados a la rehabilitación y ayuda de personas mayores o con discapacidad para conseguir una mayor autonomía. Este último tipo es en el que se centra el desarrollo de este proyecto [1].

1.1. Robótica asistencial en la Universidad Carlos III

La Universidad Carlos III de Madrid cuenta con varios robots asistenciales orientados a la rehabilitación o ayuda para obtener mayor autonomía de las personas, principalmente mayores o con algún tipo de discapacidad. A continuación se van a exponer de forma breve:

1.1.1. Robot humanoide NAO

Se trata de un humanoide de 58 centímetros de alto y cuatro kilos de peso que está siendo desarrollado por la UC3M, la Universidad de Málaga, la Universidad de Extremadura y el Hospital Universitario Virgen del Rocío de Sevilla, dentro del marco del proyecto científico Therapist [2]. El objetivo de este programa es la creación de una rehabilitación

mucho más activa y divertida que evite la desmotivación del niño, por medio de un robot capaz de indicar al paciente si está haciendo el ejercicio de forma correcta o si es necesario corregir la postura, como se puede observar en la Figura 1.1 que muestra a NAO y un paciente durante una sesión.
Dentro de este proyecto la UC3M se centra en la creación de tareas y el aprendizaje automático del robot.

Figura 1.1: Sesión de rehabilitación con NAO

1.1.2. Maggie

Es un robot social desarrollado en 2005 por el grupo de robótica de la UC3M Robotics Lab [3]. Su aspecto afable y formas redondeadas, como se muestra en la Figura 1.2, son sus rasgos más característicos.

Figura 1.2: Robot Maggie

Se trata de un robot social cuyo objetivo es interactuar con las personas de forma autónoma y directa. Aunque aún existe mucho camino para recorrer dentro del mundo de la robótica, Maggie ya es capaz de conectarse a internet, dar información, ayudar a personas con discapacidad funcional (por ejemplo en la lectura de medicamentos), leer un libro y traducirlo a cualquier idioma o jugar con el usuario [4].

1.1.3. Brazo asistencial AMOR

El grupo de robótica Robotics Lab cuenta con otros dos robot asistenciales. En este caso no son humanoides sino brazos robóticos.
El primero, que recibe el nombre de AMOR y puede apreciarse en la Figura 1.3, es especialmente apropiado para la investigación, robótica social y la educación. Más concretamente el Robotics Lab lo adquirió de la empresa ExactDynamics para orientarlo a la robótica asistencial.

Figura 1.3: Robot AMOR

Este brazo robótico cuenta con siete grados de libertad y puede llegar a extenderse hasta un metro, lo cual le hace un robot muy hábil. Además en el laboratorio se le han añadido unos sensores y cámaras que mejoran la interacción con el entorno [5].

1.1.4. Brazo asistencial ASIBOT

El segundo robot asistencial desarrollado por el Robotics Labs de la UC3M es ASIBOT [6]. Se trata un brazo robótico con 5 grados de libertad y configuración cinemática abierta y simétrica, puesto que es capaz de anclarse por ambos extremos [7].
Fue desarrollado en el año 2006 dentro del marco del proyecto RoboHealth, con el objetivo de ayudar a las personas mayores o con discapacidad (principalmente en silla de ruedas)

a conseguir una mayor autonomía en el entorno familiar. Este autómata está pensado para ayudar en tareas cotidianas como son el comer, lavarse los dientes, maquillarse o afeitarse.

RoboHealth es un proyecto multidisciplinar cuyo objetivo es la creación e introducción de robots tanto asistenciales como de rehabilitación en los hospitales, con la finalidad de mejorar el sistema nacional de salud español. Además el proyecto está orientado a aquellas personas con enfermedades crónicas y movilidad y habilidades cognitivas reducidas, que por lo tanto, tienen una gran dependencia de otras personas, por lo general de su familia. Con este trabajo se pretende dotarles de una mayor autonomía, aumentando así su calidad de vida [8].

El autómata ASIBOT, que puede observarse en la Figura 1.4, es el seleccionado en el desarrollo de este Trabajo de Fin de Grado y, por lo tanto, se va a realizar una descripción más exhaustiva del mismo.

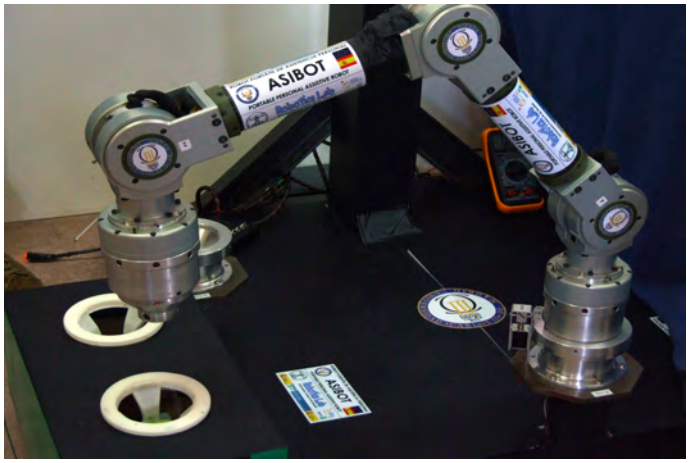

Figura 1.4: Robot ASIBOT

A continuación se presentan sus características principales:

- Como se puede apreciar en la Figura 1.4, este brazo está formado por una serie de **eslabones** (*links*) formando una cadena de 5 grados de libertad. Se trata de un robot muy ligero (tan solo 12 Kg de peso), lo cual es posible gracias a los materiales con los que está fabricado: acero para las **articulaciones** (*joints*) y fibra de carbono para los eslabones.

- Por otro lado cuenta con una gran autonomía, puesto que todos los elementos electrónicos y computacionales se encuentran integrados dentro del cuerpo. De esta forma se consigue que el robot sea fácil de transportar y de utilizar en todos los espacios de la casa, siempre que hayan sido habilitados para ello.

- Aparte de los eslabones y articulaciones ASIBOT precisa de un **anclaje mecánico** o **estación** (*docking station*), que le permite fijarse y desplazarse por el entorno, y una fuente de alimentación de 24V DC. Existen varios tipos de estaciones:

- **Anclaje fijo**: Una vez instalado no se puede desplazar. Se instala en los lugares donde sean necesarios para realizar la tarea, por ejemplo en el suelo o en la pared para ayudar a una persona a cepillarse los dientes o maquillarse o en techo y muebles para coger platos u otros elementos.
- **Anclaje móvil**: Se instala sobre raíles que permite el desplazamiento del robot mayores distancias y a mayor velocidad.
- **Anclaje en la silla de ruedas**: Base especial situada en la silla de ruedas del paciente, lo que le permite llevarlo consigo.
- **Anclaje de transición**: Permite el cambio desde la base fija a la silla de ruedas o viceversa [9] .

Gracias a estas estaciones y al bajo peso del robot, este cuenta con una capacidad de escalada que permite su movimiento entre los distintos anclajes. Además, este movimiento es posible gracias al cierre de bayoneta y los conectores especiales con forma de cono que se encuentran en cada extremo de ASIBOT y, que tienen tres funciones:

1. El extremo anclado sirve como parte de la unión con el anclaje mecánico.
2. La punta de estos conectores cónicos sirve también para la fuente de alimentación, gracias a los contactos eléctricos situados en ellos.
3. Al extremo que queda libre se pueden conectar diferentes herramientas que permiten la manipulación de objetos. Para evitar posibles riesgos los contactos eléctricos situados en la punta se encuentran ocultos.

Este movimiento entre las bases se puede observar en la Figura 1.5 y se realiza en varias etapas gracias a dichos conectores:

1. Con uno de los extremos anclado a una de las bases, el extremo libre se une a la nueva estación donde quiere ser situado el robot.
2. Una vez se realiza esta segunda unión, el extremo que estaba inicialmente conectado se desacopla dejándolo libre para realizar la acción deseada.

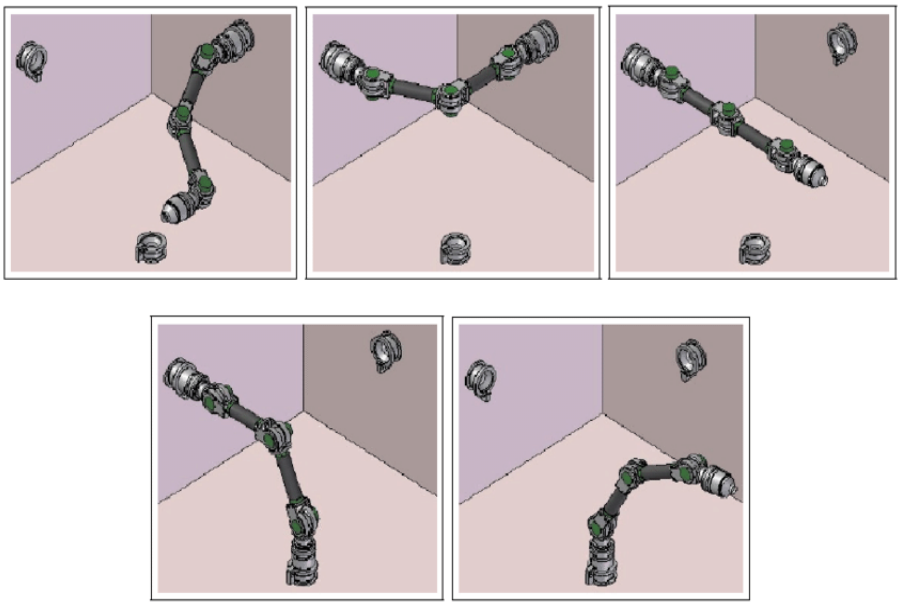

Figura 1.5: Movimiento entre los anclajes mecánicos

- En los extremos del robot es posible conectar una **garra de tres dedos**. Sin embargo, aunque esta herramienta permite sujetar y manipular muchos elementos del entorno, algunos de ellos son demasiado complejos y requieren de una precisión de la que la garra no dispone. Es por ello que se han diseñado herramientas especiales o ***Toolholder*** para estos casos. Algunos ejemplos para los que son útiles estas herramientas son para las tareas de maquillarse, cepillarse los dientes o utilizar una cuchara para comer. En la Figura 1.6 se pueden observar la garra y estas herramientas especiales.

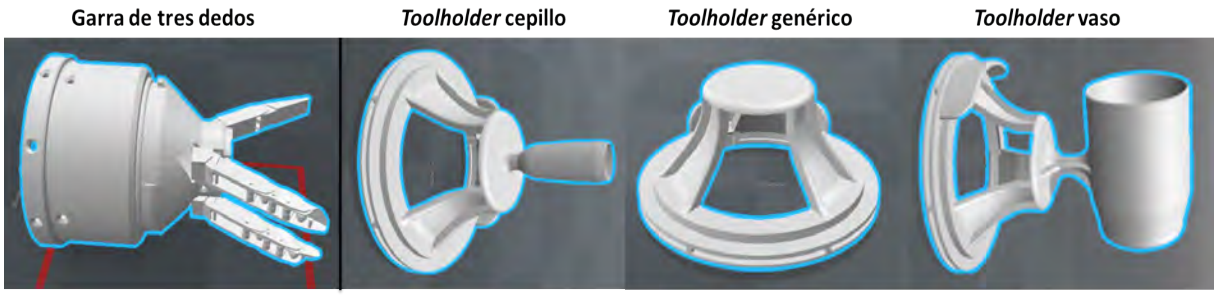

Figura 1.6: Herramientas ASIBOT

- Para realizar la interacción entre el robot y la persona que lo maneja se utiliza una **interfaz de control** por medio de una PDA, que muestra por pantalla las opciones que tiene ASIBOT y en función del grado de discapacidad se adaptan los controles para que el usuario pueda efectuar la selección:
  - **Táctil**: El usuario utiliza su dedo o un lápiz para hacer la selección.

- ***Joystick***: Permite el movimiento entre las opciones y son seleccionadas por medio de un botón.
- **Secuencia de luces con botón de selección**: Las diferentes secuencias de opciones se encienden, pudiéndose realizar diferentes elecciones.
- **Reconocimiento de voz**: Permite seleccionar y elegir por medio de comandos de voz.
- **Control de *Wii***: Permite el control dentro del *Espacio de Wii*, por medio de la alineación del *pitch* del mando con el final de manipulador de ASIBOT y el giro del mando controla el de la base [10].

En el estudio [11] realizado con cinco pacientes en el hospital de tetrapléjicos de Toledo, se determinó la utilidad y facilidad de uso de cada uno de estos elementos. Hay que destacar que ASIBOT es un asistente en la tarea, es decir, el paciente ha de ser capaz de manejarlo y reaccionar ante el entorno, es por ello que no es un robot apto para pacientes con epilepsia, deficiencia mental o visual o problemas psiquiátricos.
Así mismo las tareas se realizan en un ambiente de seguridad, pues el último contacto con él o las herramientas es realizado por el usuario, quien se acerca a realizar el movimiento final, en lugar de ser el robot el que se mueva.

Entornos de trabajo

ASIBOT se trata de un robot de asistencia orientado al ámbito doméstico. Es por esto que tanto las simulaciones como las pruebas realizadas con el modelo real se realizan en entornos dentro de la casa, en este caso el Robotics Lab, que cuenta con una cocina para hacer este tipo de pruebas en el laboratorio, como se observa en la Figura 1.7. Es por ello que el modelo que se estableció para hacer la simulación fue en el contexto de la cocina. Sin embargo, durante el desarrollo de este TFG se decidió utilizar además otro marco doméstico, el entorno de un baño, a pesar de no tener un modelo real en el que probarlo dentro del laboratorio. Adicionalmente también se decidió crear una trayectoria de rehabilitación para ASIBOT.

Figura 1.7: Cocina del laboratorio Robotics Lab

1.2. Simulación

La simulación ha sido una herramienta muy útil desde el siglo XX. Ha adquirido mayor importancia con la aparición de los ordenadores, siendo una herramienta extremadamente utilizada en casi todas las áreas de investigación y desarrollo, desde la ingeniería e informática hasta las ciencias sociales y la economía, convirtiéndose la robótica en uno de los sectores donde tiene un papel más fundamental.

La simulación consiste en la creación de un modelo por ordenador de un sistema real o teórico, con las mismas condiciones que se quieren estudiar, para poder actuar en dicho modelo y observar su comportamiento. De la misma forma permite cambiar tantas condiciones del entorno como se quieran para poder observar el comportamiento que proporcionaría en ese caso el sistema, sin necesidad de tener que realizar esos cambios en el modelo real.

Esta herramienta tiene un gran número de ventajas. Para comenzar permite la creación, desarrollo, visualización y estudio de objetos que pueden no existir, por lo que permite estudiar un entorno que ni siquiera ha sido creado para comprobar si es factible. Por otro lado, aporta mayor seguridad a un proyecto, ya que aunque el sistema se estropee o se rompa esto ocurre solo en la simulación y el entorno real queda intacto. Este mecanismo también permite el estudio en cuatro dimensiones. Es decir, es posible observar como se desarrolla el proyecto en el tiempo.
Por todos estos motivos la simulación resulta una herramienta tan útil, porque permite reducir el peligro y los daños, tanto en los trabajadores como en la maquinaria, se disminuyen los errores asociados con las vueltas atrás y posibilita el ahorro de tiempo, ya que por medio de las simulaciones se pueden hacer todas las pruebas que se quieran de un sistema. Con todas estas ventajas es sencillo entender por qué se ha convertido en una herramienta tan utilizada en robótica.

En este proyecto se han estudiado y utilizado dos simuladores: Gazebo y OpenRave, de los cuales se hablará en profundidad en el capítulo 2. Sin embargo existen otros que se describen someramente:

1. **ARGoS**: es un simulador robótico que incluye múltiples modelos físicos, desarrollado dentro el proyecto Swarmanoid. Permite la simulación de robots complejos a cualquier nivel de eficiencia y es posible adaptarlo según las necesidades del usuario por medio de *plug-ins*. Fue la herramienta principal en varios proyectos europeos:*ASCENS*, *H2SWARM*, *E-SWARM* y *Swarmix* [12].

2. **Coppelia Robotics**: Plataforma virtual de experimentación con robots (*V-rep*), que permite la creación, composición y simulación de cualquier robot. Este simulador además permite el desarrollo de los controladores de cada robot en cualquiera de estos lenguajes de programación: C/C++, Python, Java, Lua, Matlab u Octave. Actualmente, V-rep es utilizado en múltiples campos, tales como la simulación de plantas automatizadas, el desarrollo de algoritmos, control remoto y comprobaciones de seguridad entre otros [13].

3. **Webots**: Se trata de un simulador disponible para todos los sistemas operativos, que cuenta con una gran red de usuarios, lo que permite la comunicación y ayuda entre los mismos. Además la información referente al programa es constantemente actualizada. Asimismo, cuenta con una simulación dinámica y de colisiones, interfaz tanto para Matlab como para Ros y una amplia librería de robots a su disposición [14].

1.3. Motivación del proyecto

Como se ha mencionado anteriormente, los robots están adquiriendo mayor protagonismo en la vida de las personas. Existen autómatas utilizados en las fábricas y en la vida doméstica, poco a poco se está introduciendo un mayor número de ellos en los hospitales para ayudar en las rehabilitaciones o con las operaciones quirúrgicas(como el Robot Da Vinci [15]) e incluso, se están desarrollando robots con una finalidad social (como el robot Maggie que se mencionaba en el punto 1.1.2).

Uno de los campos donde los robots pueden ser de gran utilidad y tener un importante futuro es en el campo de la robótica asistencial, es decir, robots orientados a la ayuda de personas con discapacidad o ancianos. Actualmente la esperanza de vida es mayor, lo que también significa que existe una mayor cantidad de gente anciana que en muchos casos requieren de asistencia de sus familiares o cuidadores sociales. Existe además otro grupo, las personas con discapacidad, que precisan de este tipo de asistencia pues no son capaces de ser completamente autónomos en todos los aspectos de su vida. Esta falta de autonomía podría disminuirse por medio de robots asistenciales, que permitiesen a las personas dentro de estos grupos realizar tareas cotidianas sin la ayuda de sus cuidadores. Ganando de esta forma en confianza e independencia, pues serían capaces de volver a realizar tareas del día a día por si solos.

Dentro de este marco de la robótica asistencial se desarrolló ASIBOT, con el objetivo de poder devolver, en cierta medida, esa independencia a aquellas personas que lo necesiten.

ASIBOT fue creado hace once años, por lo que su modelado para simulación se definió en OpenRave. Sin embargo, este programa está comenzando a quedar obsoleto y a perder terreno frente al modelo de Gazebo, que cuenta con mayores prestaciones, pues es posible definir una mayor cantidad de elementos del entorno y característas, como se detallará en el capítulo 2.
Ante este crecimiento del simulador Gazebo, el Robotics Lab consideró necesaria la migración de todos los archivos relacionados con ASIBOT desde OpenRave a Gazebo, con el fin de poder trabajar en un entorno de simulación que ofrece un mayor número de ventajas y es el modelo con el que se trabaja actualmente dentro de la robótica.

1.4. Objetivo del proyecto

El objetivo del Trabajo de Fin de Grado, que se presenta a continuación, es conseguir crear dos trayectorias para el robot de asistencia ASIBOT por medio de MoveIt. También si es posible, visualizarlas en el entorno de simulación Gazebo. Para ello el trabajo queda dividido en dos partes bien diferenciadas:

- **Conversión de OpenRave a Gazebo**: Transformación de los archivos .xml utilizados en OpenRave a .sdf y .urdf, modelos utilizados en Gazebo.
- **Creación de trayectorias**: Para ello se utilizarán los programas MoveIt, que es un software utilizado en aplicaciones robóticas y Rviz, visualizador 3D asociado a ROS. Adicionalmente, se probarán las mismas dentro del entorno de simulación de Gazebo.

1.5. Estructura

Este Trabajo de Fin de Grado sigue la siguiente estructura:

En el capítulo 2 se hace una introducción al entorno de trabajo y posteriormente se realiza un estudio y comparación entre los dos sistemas de simulación con los que se ha trabajado en este proyecto.

En el capítulo 3 se explica la transformación de los archivos de OpenRave a aquellos utilizados por Gazebo.

En el capítulo 4 se estudia la creación de las trayectorias por medio del asistente de MoveIt y del programa Rviz.

En el capítulo 5 se presentan los resultados obtenido en Rviz mediante la creación de las trayectorias y se realiza un análisis crítico y conclusiones sobre los mismos.

En el capítulo 6 se proponen mejoras sobre el trabajo realizado.

En el apéndice A se presenta el marco regulador del proyecto.

En el apéndice B se hace referencia al impacto socio-económico y al presupuesto del trabajo.

En el apéndice C se presentan los códigos relevantes para el proyecto.

Capítulo 2

Estado del arte

En estas últimas décadas, la automatización de los hogares ha estado reducida a los electrodomésticos, como son los lavavajillas, hornos o lavadoras, entre otros. Sin embargo, poco a poco se ha ido produciendo una automatización de las viviendas introduciéndose alarmas, sensores, luces automáticas, etcétera, siendo la tecnología de inteligencia ambiental [16] el siguiente paso en este desarrollo. Esta tecnología busca la integración de todos estos elementos con el fin de recopilar información sobre los hábitos de la casa, mejorando la eficiencia de estos componentes y reduciendo el esfuerzo humano.
Por otro lado, hay que recalcar que, aunque estos elementos queden integrados, son estáticos, es decir, se instalan en la casa y no cambian su lugar, ni interactúan con el entorno. Sin embargo, se han empezado a introducir otros elementos más autónomos en contraposición, como es el ejemplo de las aspiradoras automáticas como la *roomba*. El objetivo de estos elementos con una mayor autonomía, es precisamente el conseguir que realicen tareas del hogar sin o con muy poca supervisión humana, o bien ayudar a la persona a poder realizar esa actividad con un menor esfuerzo. Para ello es necesario dotarlos de una mayor autonomía en los movimientos, es decir, tienen que dejar de ser estáticos y tienen que tener un mayor números de sensores que les permitan recoger más información sobre el entorno. Dentro de este marco de autómatas aparece el robot portátil de asistencia a discapacitados ASIBOT.

Esta creación de robots más autónomos que se pretenden introducir en los próximos años, requiere también la aplicación de modelos de simulación que permitan hacer las pruebas necesarias para asegurar que el autómata es capaz de operar en el entorno deseado de una forma eficiente y segura para los usuarios.

Por supuesto, el uso de simulaciones lleva siendo utilizado desde mucho antes del desarrollo de estos robots. La mayoría de los campos de la ciencia e ingeniería utilizan este método para obtener resultados y comprobaciones mucho más rápidamente y de forma más barata que desarrollando un experimento físico.

Sin embargo, antes de comenzar a describir los distintos simuladores que se utilizan en este Trabajo de Fin de Grado y sus características, es necesario explicar, aunque sea brevemente, el entorno de trabajo: ROS.

2.1. ROS: *Robot Operating System*

Se trata de un entorno informático de trabajo libre, flexible y colaborativo que permite el desarrollo de software robótico. Cuenta con un amplio número de herramientas, librerías y convenciones que buscan simplificar la creación de robots complejos en las diferentes plataformas robóticas.
Fue creado desde cero en el 2000 dentro de la Universidad de Stanford, involucrando inteligencias artificiales como la *STanford AI Robot (STAIR)* y el programa *Personal Robots (PR)*, por medio de prototipos flexibles y dinámicos de sistemas de software orientados

a la robótica. Sin embargo, no fue hasta 2007 cuando Willow Garage facilitó los recursos para extenderlo y crear versiones completamente funcionales. El espíritu colaborativo es la base de ROS, el cual está diseñado para que diferentes grupos cooperen y construyan sobre el trabajo que otros han hecho.

La comunidad de ROS está creciendo muy rápidamente y cada vez son más las características y aplicaciones que hacen que se elija este modelo frente a otras plataformas robóticas. Es por esto que, aunque dentro de este Trabajo de Fin de Grado no se trabaja directamente manipulando los elementos que definen a ROS, s se ha visto necesario explicar brevemente sus características y estructura, pues es la plataforma que está detrás de todos los programas utilizados para el desarrollo de este proyecto. La arquitectura de ROS puede ser dividida en tres secciones o niveles: sistema de archivos, computación gráfica y nivel comunitario, que se explican a continuación.

Antes de empezar con dicha explicación es necesario aclarar que existen varias versiones de ROS, por lo tanto existen ciertas órdenes o comandos que tienen que ser adapatados según cuál esté siendo utilizada. En el caso de este TFG se utilizó la décima versión oficial *ROS Kinetic Kame*. Además, es necesario señalar que todos los comandos utilizados a lo largo de este trabajo se lanzan desde la terminal de Ubuntu [17].

2.1.1. Nivel sistema de archivos (*Filesystem level*)

En esta subsección se explica cómo está formado ROS internamente, la estructura de sus carpetas y los archivos mínimos necesarios para poder funcionar.

La Figura 2.1 muestra las diferentes carpetas en las que está dividido. Dentro de cada carpeta existen diferentes archivos que describen su funcionalidad:

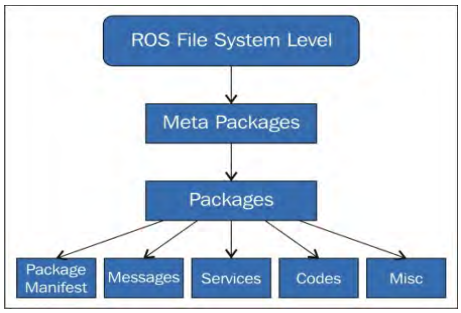

Figura 2.1: Jerarquía del sistema de archivos

- **Meta paquetes (*Metapackages*)**: Son una agrupación de paquetes. Sin embargo, el único archivo que hay dentro de ellos es el *package.xml* y es utilizado para hacer referencia a los distintos paquetes dentro del grupo.
- **Paquetes (*Packages*)**: Es considerada la estructura mínima que permite crear un programa en ROS. Está formado por diferentes archivos y carpetas, como el *package.xml*, cuya presencia es obligatoria dentro del paquete y proporciona información

sobre el mismo (nombres, dependencias, etcétera).
Dentro de esta carpeta también se pueden encontrar librerías, ficheros de ejecutables, archivos fuente (en la carpeta src/) y el archivo *CMakeLists.txt*, que es el archivo CMake utilizado para compilar el programa.
ROS proporciona multiples comandos que permiten moverse entre los paquetes, crearlos y modificarlos. Todos ellos se pueden encontrar dentro de la página oficial de ROS [17] y por lo tanto, en este trabajo solo se mencionarán aquellos que sean utilizados.

- **Manifiesto del paquete (*Package manifest*)**: Es gestionado por el *package.xml* y, como se menciona en el punto anterior, proporciona información sobre el paquete.
- **Mensajes (*Message type* o *msg*)**: La información que se comparte entre los distintos procesos se hace por medio de los mensajes. Además describen el valor de los datos publicados por los nodos, procesos definidos en el punto 2.1.2.
  Los mensajes se componen de dos partes principales: el campo (*field*), que define el tipo de dato que se transmite (enteros, cadenas de caracteres o nuevos datos establecidos por el usuario) y las constantes (*constants*), que determinan el nombre del campo. La descripción de estos mensajes se encuentra dentro del archivo: *my_package/msg/MyMe- ssageType.msg*.
- **Servicios (*Service type* o *srv*)**: Define las solicitudes y respuestas de las estructuras de datos para los servicios proporcinados por los procesos de ROS.
  Se construye encima del formato de mensajes para permitir la comunicación entre los distintos nodos y su descrición se puede encontrar dentro de *my_package/srv/MyServiceType.srv*.

El **entorno de trabajo (*workspace*)** es la carpeta donde se encuentran y compilan los paquetes y se editan los archivos fuente. Resulta muy útil cuando se tienen varios paquetes que se desean compilar a la vez.
En el caso de este proyecto el entorno principal de trabajo es denominado "*catkin_ws*".

2.1.2. Nivel computación gráfica (*Computation graph*)

En este segundo nivel se produce la comunicación entre los distintos procesos y sistemas, por medio de una red donde todos los procesos están conectados y a la que cualquier nodo puede acceder, tanto para interactuar con otros, como para ver la información que se está mandando y transmitir datos a la red. Los elementos que conforman esta red se pueden ver en la Figura 2.2 y son descritos a continuación:

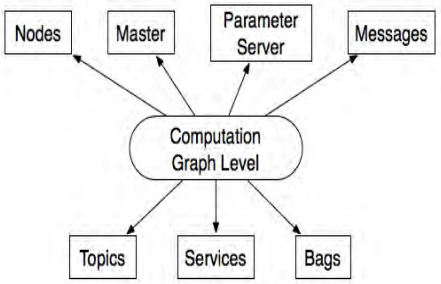

Figura 2.2: Nivel de computación gráfica

- **Nodos (*Nodes*)**: Son los procesos donde se hace la computación, es decir, son ejecutables que pueden comunicarse con otros procesos utilizando temas, servicios o el servidor de parámetros.
  El uso de nodos hace que el sistema sea más sencillo, pues permite separar los códigos de las funciones.

- **Maestro de ROS (*ROS Master*)**: Es el encargado de manejar toda la comunicación, sin él esta sería imposible. Permite que los nodos individuales de ROS se encuentren unos a otros, por medio del registro de nombres y la búsqueda de servicios.

- **Servidor de parámetros (*Parameter server*)**: Permite almacenar datos en una localización central mediante el uso de llaves. Los nodos lo utilizan para guardar y extraer parámetros durante el tiempo de ejecución.

- **Mensajes**: Como han sido explicados en el punto 2.1.1 (nivel anterior), son la forma en la que los nodos se transmiten la información.

- **Temas (*Topics*)**: Son buses utilizados por los nodos para transmitir datos y pueden ser utilizados sin una conexión directa entre los nodos.
  Aunque un mismo tema puede tener varios suscriptores y editores, hay que tener en cuenta que su nombre debe ser único para evitar problemas y confusiones entre varios temas a la vez.

- **Servicios (*Services*)**: Ofrecen la oportunidad de interactuar con los nodos y recibir una respuesta, operaciones que no se pueden hacer con los temas.

- ***Bags***: Tipo de formato para almacenar datos. Es un archivo creado por ROS para guardar la información de los mensajes, temas y servicios.

2.1.3. Nivel comunitario *(Community level)*

Se trata del útimo nivel de ROS y es el que permite a diferentes comunidades de usuarios intercambiar sotware y conocimientos. Incluye tres grandes grupos:

- **Distribución de ROS**: Diferentes colecciones de meta paquetes que se pueden instalar y hacen más sencillo conseguir el sotfware.
- **Repositorios de ROS**: Red federal donde las diferentes instituciones pueden desarrollar y compartir el software de los componentes de su robot.
- **ROS Wiki**: Foro con información documentada sobre ROS, entre ella múltiples tutoriales para aprender a manajerlo.

[18]

2.2. Simuladores robóticos: OpenRave y Gazebo

En el capítulo 1 se hizo una breve introducción a la simulación y a distintos ejemplos de simuladores que existen actualmente. La simulación consiste en crear un modelo virtual imitando a la realidad y esto es precisamente lo que se pretende conseguir en este TFG por medio del estudio de dos simuladores: Gazebo y OpenRave.

2.2.1. OpenRave

OpenRave (*Open Robotics Automation Virtual Environment*) es un software de código abierto y multiplataforma para la simulación de robots. Integra servicios de análisis y simulación cinemática y geométrica [19].

La característica y ventaja principal de este simulador es su arquitectura de *plugins* (fragmentos de software que añaden características adicionales al programa), que permite conectarse fácilmente con otras plataformas y la comparación entre distintos miembros de la robótica para compartir e intercambiar algoritmos y códigos. Además cuenta con un gran número de algoritmos utilizados en la creación de trayectorias, permite depurar los componenes mientras el sistema está en marcha sin necesidad de reiniciarlo por completo y permite la creación, almacenaje y retirada de elementos para la planificación.

En definitiva, se trata de un sotfware que pretende simplificar la simulación de robots en escenarios complejos utilizando la cinemática, control de robots, detección de colisiones, la dinámica y scripts del entorno (lenguaje informático formado por un conjunto de comandos, que puede ser ejecutado sin compilación) [20].

2.2.2. Gazebo

Gazebo es un simulador 3D utilizado para simular de forma precisa y eficiente diferentes robots en complejos entornos tanto interiores como exteriores [21].

Principalmente se utiliza para diseñar robots, evaluar algoritmos robóticos y realizar evaluaciones regresivas en entornos realistas. Para ser capaz de hacer todo eso Gazebo

cuenta con una gran simulación dinámica, con acceso a máquinas físicas de alto rendimiento, avanzados gráficos en 3D, una amplia colección de robots a disposición del usuario, a la vez que permite la creación de uno propio utilizando un archivo en SDF. También posibilita la generación de datos de sensores y de *plugins* para el robot y control del entorno y sensores. Por último cuenta con una amplia lista de comandos que facilitan la simulación y su control.

En pocas palabras, se trata de un software gratuito, con gráficos de alta calidad, una programación cómoda e interfaces gráficas, que se encuentra dentro de una gran comunidad que permite el intercambio de conocimiento entre sus usuarios [22].

2.3. Creación de entornos: Comparación entre Gazebo y OpenRave

En esta última parte del capítulo se van a analizar y estudiar las similitudes y diferencias que existen entre los simuladores Gazebo y OpenRave para poder realizar la conversión entre ambos.

Se comienza con unas nociones sobre el entorno y la creación del mismo en los dos programas, para continuar, estudiando más en profundidad las conversiones entre ellos. Hay que recalcar que se analizarán aquellos aspectos que resulten más útiles para este Trabajo de Fin de Grado, pues ambos simuladores son muy completos y la explicación total supondría una gran extensión.

A la hora de realizar el estudio se analizan en primer lugar las características de OpenRave para buscar posteriormente el paralelismo con Gazebo. Esto se debe a que el punto de partida de este proyecto son los archivos definidos para OpenRave, por lo tanto es necesario comprenderlos primero, para poder realizar una conversión adecuada y eficiente a Gazebo.

2.3.1. Definición del entorno

El primer paso cuando se trabaja en simulación es decidir y definir como va a ser el entorno de trabajo, por ello en esta subsección se va a explicar qué es un entorno, por qué elementos está formado, sus características y su estructura básica.

El área por la que se mueve el robot es lo que se define como **entorno** y, dentro del mismo, se incluyen los objetos y características físicas que influyen en la conducta del robot.

Todos los objetos definidos dentro del entorno son considerados **modelos**. Un modelo puede ser desde un objeto con una forma sencilla a un robot complejo. Sin embargo, modelo también puede ser utilizado para definir sensores o características estáticas como pueden ser el suelo o el sol.

Un modelo, a su vez, es un compuesto de **cuerpos cinemticos** (***links***), que son definidos mediante **elementos visuales** y de **colisión** (***visual and collision elements***), es decir, por medio de geometrías, **articulaciones** (***joints***), que realizan la unión entre los eslabones y ***plugins*** [23].
En las Figuras 2.3 y 2.4 se puede encontrar un ejemplo de entorno de simulación en Gazebo y OpenRave respectivamente.

Figura 2.3: Entorno en Gazebo

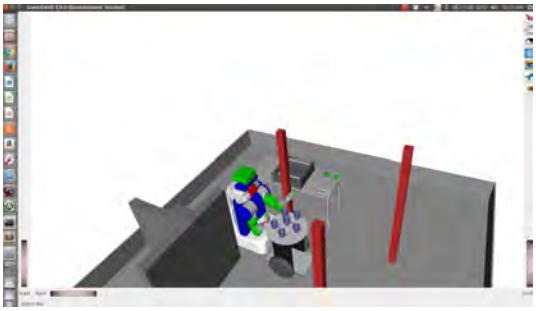

Figura 2.4: Entorno en OpenRave

2.3.2. Formato de lenguaje

Cada modelo de un entorno está definido dentro de un archivo, que describe sus características y los elementos que lo conforman. Para esta descripción se utiliza un lenguaje que varía en función del simulador que se está utilizando. Por ello, es necesario explicar primero de que lenguajes se sirven estos dos simuladores antes de poder realizar la comparación entre las definiciones de los elementos.

Ambos programas tienen como base el lenguaje XML, en el caso de Gazebo desarrolla a partir del mismo un nuevo formato, y en el de OpenRave, lo personaliza para las necesidades del simulador. Por esta razón se explica primero el formato base y después se ahonda en los lenguajes propios de cada simulador.

Lenguaje XML

El lenguaje XML (*Extensible Markup Lenguage*) es un formato de texto sencillo y muy flexible, que permite el intercambio de una gran variedad de datos dentro de la Web [24].

Este lenguaje se utiliza para describir datos, es decir, es una forma flexible de crear formatos de información y de compartir estructuras de datos de forma electrónica, pero no genera resultados, se trata de información definida por medio de etiquetas y para que se mande, reciba o muestre es necesario el uso de un software [24].

XML no es un lenguaje de programación sino un conjunto de reglas para ordenar la información, que se define por medio de **etiquetas**, palabras escritas entre los símbolos

"<" y ">" y **atributos**, que se definen como `name=''value''`. Otras características de este lenguaje de descripción son que solo utiliza estas etiquetas para delimitar los datos, y su interpretación depende completamente de la aplicación que lo ejecute. Además, es muy estricto con sus reglas, ya que una etiqueta no definida o un atributo sin comillas hacen el archivo inservible.
Esa libertad de interpretación hace que los lenguajes derivados del XML hereden su sintaxis, es decir, las etiquetas y atributos, pero los redefinan para su uso concreto [25].

Lenguaje de OpenRave

OpenRave utiliza el lenguaje XML personalizado en sus archivos para la definición de entornos y robots, aunque adicionalmente también se pueden definir por medio de archivos COLLADA [26]. A continuación se explican ambos formatos.

- **XML**
  El formato que utiliza OpenRave permite definir tanto los robots como su entorno y además es lo suficientemente flexible como para permitir la unión con otros archivos XML o en formato .vrml o .iv, de forma que se puedan importar modelos dentro del entorno.

  Los diferentes elementos del entorno se definen por medio de distintas etiquetas y archivos que se estudian en la sección 2.3.3 [26].

- **COLLADA**
  OpenRave también permite este formato para la definición de robots y le añade su propio conjunto de extensiones. Este formato se sigue pudiendo utilizar para especificiar la información relacionada con el autómata y su entorno, ya que permite definir sensores, parámetros de planificación, manipuladores e información de colisiones [27].

  Existen dos formatos para guardar los archivos del tipo COLLADA: el .dae que guarda el XML puro y el .zae que almacena el XML comprimido. Por tema de espacio la mayoría de robots se guardan en este formato [28].

Lenguaje de Gazebo

En el caso de Gazebo existen tres formas diferentes de lenguaje para definir los archivos del entorno y los modelos: URDF, SDF y Xacro, todos ellos derivados del lenguaje XML.

- **URDF: Universal Robot Description Format**
  Es un lenguaje XML utilizado por ROS para describir todos los elementos de un robot. Pero para poder utilizarlo correctamente en Gazebo es necesario añadir algunas etiquetas específicas para la simulación.

Sin embargo, a pesar de que el URDF es un lenguaje útil y estandarizado en ROS, no se ha ido actualizando para poder adaptar las necesidades robóticas, y por lo tanto le faltan varias características.
Actualmente solo es capaz de definir las propiedades dinámicas y cinemáticas de un robot aislado, es decir, no puede definir su posición dentro de un entorno o propiedades físicas como la fricción. Además es incapaz de definir elementos que no sean robots, como por ejemplo la luz solar.

Para solucionar estas limitaciones se creó el lenguaje SDF [29].

- **SDF: Simulation Description Format**
  Fue creado para Gazebo basándose en el lenguaje XML y con el fin de resolver las deficiencias del modelo URDF. Permite hacer una descripción mucho más detallada del entorno y todos los modelos existentes en él, además de que resulta fácil añadir y modificar elementos dentro del mismo y resulta un formato autodescriptivo [29].

  Con el SDF es posible definir y representar diferentes propiedades físicas, de iluminación, el terreno, objetos dinámicos y estáticos y robots articulados con múltiples sensores y actuadores [30].

  El código definido en un archivo .sdf se encuentra dentro de una etiqueta sdf como la mostrada en la Figura 2.5. Hay que tener en cuenta, que el número de la versión puede ser diferente del de la versión de *libsdformat*, repositorio que contiene las librerías en C++ que se utilizan para analizar el archivo .sdf [31].

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <!--Codigo del modelo-->
</sdf>
```

Figura 2.5: Etiqueta que define el encabezado y versión de un archivo .sdf

- **Xacro**
  Es un lenguaje XML basado en macros, que son plantillas para la ejecución de una secuencia de instrucciones de programación. Este formato permite la creación de archivos .xml más cortos y comprensibles. Debido a esta característica es un formato bastante utilizado en la descripción de robots [32].

  Existe un comando de ROS que permite la transformación de ficheros .xacro a .urdf: "`$ rosrun xacro xacro.py model.xacro > model.urdf`", el cual puede ser útil en caso de tener que utilizar programas que necesiten un .urdf como se ve en el capítulo 4. Sin embargo, la transformación inversa, es decir, de .urdf a .xacro no es una transformación que se pueda hacer por medio de un comando y además, el proceso utilizado depende de la plataforma que se esté utilizando [33].

2.3.3. Definición de archivos

XML permite que se relacionen fácilmente varios archivos .xml. Por lo tanto, para casos complejos es posible definir el entorno, modelos y robots en archivos separados que luego se relacionen entre sí.

La forma de definir el archivo cambia según aquello que se quiera definir en él, por eso se explican a continuación las etiquetas y atributos que son utilizados para definir los diferentes modelos.

Archivos que definen el entorno

A la hora de realizar una simulación es imprescindible tener un archivo que defina el entorno. Dentro de este fichero se definen características físicas, posiciones de los elementos presentes en la simulación: robots, objetos dinámicos y estáticos, sensores, etcétera. Además permiten el control de diferentes parámetros de la simulación como es el tiempo.

- **OpenRave**
  Dentro de OpenRave los ficheros de entorno se definen por medio de la extensión "**.env.xml**" y dentro del archivo todo el código queda encuadrado dentro de la etiqueta **`Environment`**, como se puede ver en la Figura 2.6.

```
<Environment>
  <!-- Definición de los elementos del entorno-->
</Environment>
```

Figura 2.6: Etiqueta que define el archivo de entorno en OpenRave

- **Gazebo**
  En el caso de Gazebo los archivos que definen el entorno tienen una extensión "**.world**" y engloban el código dentro de la etiqueta **`world`**. Además, esta etiqueta requiere la elección de un nombre para el entorno que se escribe utilizando el atributo `name='nombre'`. la Figura 2.7 muestra la estructura básica de este archivo.

```
<sdf version='1.6'>
  <world name='asibot_world'>
    <!-- Definición de los elementos del entorno-->
  </world>
</sdf>
```

Figura 2.7: Etiqueta para definir el archivo de entorno en Gazebo

Cuerpos cinemáticos o modelos

Como ya se ha mencionado, estos archivos son utilizados para definir cuerpos cinemáticos o robots. La razón para definirlos en archivos diferentes al entorno es porque

pueden llegar a ser bastantes complejos. Además, de esta forma también se simplifica el fichero del entorno teniendo simplemente que llamar al archivo .xml que define el cuerpo que se desea introducir.

- **OpenRave**
  Los modelos cinemáticos dentro de OpenRave se definen por medio de la etiqueta **`Kinbody`**, dentro de la cual se definen todos los elementos que conformen el cuerpo y sus características y, tienen la extensión "**.kinbody.xml**". En la Figura 2.8 se puede observar la estructura de este archivo.

```
<KinBody name="uc3m-asibot">
  <!-- Definición del modelo-->
</KinBody>
```

Figura 2.8: Etiqueta para definir un modelo en OpenRave

  Para añadir el modelo al archivo del entorno solo es necesario añadir en el mismo la etiqueta **`Kinbody`**, seguido del atributo **`file`** y la dirección al archivo del modelo, como se observa en la Figura 2.9.

```
<KinBody file="./docking.kinbody.xml">
        <!--Transformaciones aplicadas al modelo-->
</KinBody>
```

Figura 2.9: Etiqueta para incluir un modelo en el archivo de entorno en OpenRave

- **Gazebo**
  La definición en Gazebo viene incluida dentro de la etiqueta **`model`**, siendo necesario añadir el nombre del modelo por medio del atributo `name='nombre'` y con una extensión "**.sdf**". Sin embargo, en el caso de Gazebo la definición de los archivos es un poco más complicada que en OpenRave.
  Cada modelo tiene que estar definido dentro de una carpeta donde tienen que encontrarse los siguientes archivos:

  - model.sdf: Contiene la descripción del archivo.
  - model.config: Engloba los metadatos del modelo: nombre del modelo, descripción del mismo o autor, entre otros.
  - meshes: Directorio con los archivos 3D, que definen las mallas que conforman los objetos, en formato .stl o COLLADA.
  - material: Directorio que contiene las texturas y *plugins* asociados al modelo.

  Los dos últimos puntos no son imprescindibles, pues la dirección a la carpeta con las mallas puede ser definida dentro del modelo, al igual que las texturas.
  La Figura 2.10 muestra un ejemplo de la definición de este tipo de archivos.

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <model name='asibot'>
    <!-- Definición del modelo-->
  </model>
</sdf>
```

Figura 2.10: Etiqueta para definir un modelo en Gazebo

En el caso de Gazebo, para incluirlo en el fichero del entorno se necesitan dos etiquetas: primero la etiqueta **`include`**, que indica que se va a añadir un fichero externo, y segundo la etiqueta **`uri`**, en la que se indica la dirección del modelo.sdf. El código de la Figura 2.11 indica cómo incluir archivos de forma correcta.

```
<include>
  <uri>model://model</uri>
</include>
```

Figura 2.11: Etiqueta para incluir un modelo en el archivo de entorno en Gazebo

Robots

Los robots son un tipo de modelo especial, pero su definición e inclusión en el archivo del entorno es muy similar a la utilizada para los modelos definidos en el punto 2.3.3.

- **OpenRave**
  En este caso, la mayor diferencia es que el archivo se define con una extensión "**.robot.xml**" y mediante la etiqueta **`robot`**, como se muestra en la Figura 2.12.

```
<robot name="modelo">
 <!--Definición del robot-->
</robot>
```

Figura 2.12: Etiqueta para definir un robot en OpenRave

Para incluirlo dentro del entorno se hace igual que con los modelos pero por medio de la etiqueta **`robot`**, seguido de los atributos `name=''nombre_robot''` y `file=''dirección archivo.robot.xml''`, como se puede ver dentro de la Figura 2.13.

```
<Robot name="Asibot" file="./asibot_p_mot.robot.xml">
      <!-- Cambios en la posición del robot-->
</Robot>
```

Figura 2.13: Etiqueta para incluir un robot en el archivo de entorno en OpenRave

- **Gazebo**
  En Gazebo la diferencia principal entre los robots y los modelos es que estos primeros se pueden definir tanto en formato .sdf como .urdf, aunque la estructura de la carpeta se mantiene igual.
  En el caso del model.sdf el robot se sigue definiendo por medio de la etiqueta **`model`**, como se mostraba en la Figura 2.10. Sin embargo, en el model.urdf la etiqueta utilizada es **`robot`**, como se puede observar en la Figura 2.14.

```
<?xml version="1.0" ?>
<robot name="asibot">
 <!-- Definición del robot-->
</robot>
```

Figura 2.14: Etiqueta para definir un modelo dentro de un archivo .urdf

La inclusión en el fichero del entorno se sigue realizando referenciando el archivo del robot como se ha indicado con `model` en el punto 2.3.3.

Mallas tridimensionales

Por lo general, los modelos y los robots están compuestos de piezas con una geometría compleja, es decir, diferente de cajas o esferas. Por lo tanto, necesitan ser definidas de otra manera. Para este caso se crean las mallas tridimensionales, que son modelos de la pieza en 3D. Estos modelos se pueden diseñar con programas de modelado 3D y el formato en el que se definen depende del simulador que se está utilizando.

- **OpenRave**
  OpenRave admite dos formatos para la definición de mallas 3D: Open Inventor(.iv) y VRML(.wrl).

  - *Formato Open Inventor*
    El formato .iv es la extensión utilizada por el programa Open Inventor, que es un software orientado al desarrollo de objetos en 3D, el cual, por medio de un conjunto de herramientas, ofrece soluciones a problemas gráficos de programación dentro del ámbito de la medicina, ingeniería y la minería de gas y aceite [34].
    Este modelo es utilizado para describir nodos y campos, siguiendo un formato especial denominado *syntax* (forma de describir el lenguaje tanto humano como de programación), que sigue un conjunto de normas que no se detallarán por simplicidad del documento, puesto que no es necesario conocerlas para ser capaz de realizar la transformación que se busca en este TFG [35] [36].
  - *Formato VRML*
    El VRML (Virtual Reality Modeling Lenguage) es un lenguaje tridimensional e interactivo orientado a la modelación y visualización de objetos diseñado para su uso en páginas web. Sin embargo, actualmente ha sido sustituido por el X3D [37].

Para incluir estas mallas dentro del archivo se hace por medio de las etiquetas **`render`** o **`data`**, dependiendo de la función que cumplan, la cual se explicará más adelante en la sección 2.3.4. La Figura 2.15 indica cómo añadir una malla con ambas etiquetas.

```
<data>./ivs/ensam01sn.iv</data>
<render>./ivs/ensam01sn.iv</render>
```

Figura 2.15: Etiqueta para incluir una malla en OpenRave

- **Gazebo**
  Los modelos definidos para Gazebo aceptan dos formatos: archivos STL o COLLADA.

  - *Formato STL*
    El formato STL (STereoLithography) fue creado por *3D Systems* y es muy utilizado en el prototipado rápido, impresión 3D y en la fabricación asistida por ordenador. Este formato se caracteriza por aproximar las superficies de los modelos por medio de triángulos. De esta forma, cuanto más compleja es la figura mayor es el número de triángulos necesarios para definirla.
    A día de hoy casi todos los programas de modelado en 3D son capaces de generar este formato. Sin embargo, hay que diferenciar entre las dos opciones que existen: el archivo .stl ASCII y el .stl binario. Ambos siguen definiendo la estructura por medio de triángulos, pero la mayor difencia es que el binario resulta más compacto. Esta característica será tomada en cuenta en el posterior desarrollo de este TFG [38].
  - *Formato COLLADA*
    COLLADA (COLLAborative Design Activity) es un formato basado en XML, que permite la creación de piezas en 3D [39].
    Permite definir de forma sencilla geometrías, sombras y efectos, propiedades físicas, cinemática y animación. Además este formato puede ser utilizado por distintas plataformas que trabajen con objetos tridimensionales [40].

  Para incluir estas mallas 3D en el archivo del modelo se utiliza la etiqueta **`uri`**, escribiendo dentro la dirección donde se encuentra el fichero, como se indica en la Figura 2.16.

```
<uri>model://pioneer2dx/meshes/chassis.dae</uri>
```

Figura 2.16: Etiqueta para incluir una malla en Gazebo

Cuando se incluye una malla tridimensional o varias, es posible que el tamaño no coincida o no sea el que se desea. Para esos casos se puede definir la propiedad de la **escala**, que permite ajustar la dimensión del cuerpo.

- **OpenRave**
  En OpenRave todo lo que hay que hacer para añadir una escala es colocar el factor de escalado tras la dirección al archivo de mallas dentro de la etiqueta **`render`**.

- **Gazebo**
  En Gazebo la escala se indica por medio de la etiqueta **`scale`**, indicando el factor de escalado en los tres ejes, justo debajo de la etiqueta **`uri`**.

- **Comparación de código**

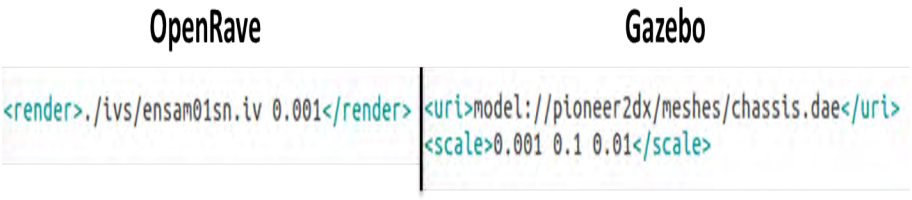

Figura 2.17: Comparación de código de escalas

2.3.4. Elementos del entorno

En la subsección 2.3.3 se ha explicado como se define el entorno, los modelos y robots en los diferentes modelos de simulación. En esta subsección se va a analizar y comparar los elementos dentro de estos archivos anteriores.

Para llevar a cabo este análisis se explica brevemente la función del elemento que se está analizando, posteriormente se explica como se definen, primero en OpenRave y luego en Gazebo, y por último se compararán fragmentos de código para que se puedan apreciar estas diferencias y similitudes de forma más clara.

Por último, antes de comenzar con el estudio se requiere explicar la jerarquía que existe dentro del archivo. Por lo general el **entorno** está formado por una serie de **modelos** o **robots**. Estos a su vez están formados por **eslabones** o cuerpos sólidos denominados **links**, que pueden estar unidos mediante **articulaciones** o **joints**. Dentro de cada cuerpo se define la geometría del mismo.

Cuerpos sólidos o links

El link es la unidad básica que define un modelo.

- **OpenRave**
  En OpenRave un cuerpo sólido se define mediante la etiqueta **`Body`**, que admite los atributos **`name`** y **`type`**.

- **Gazebo**
  En el caso de Gazebo se utiliza la etiqueta **`link`** que contiene solo el atributo **`name`**, el atributo **`type`** definido en OpenRave se establece como otra etiqueta dentro del cuerpo.

  En ambos caso dentro de las etiquetas se definen las geometrías y transformaciones asociadas al cuerpo.

- **Comparación de código**

Figura 2.18: Comparación de código de cuerpos o *links*

Geometrías

Cada cuerpo está compuesto por una serie de geometrías que se definen dentro de la etiqueta de `link` estudiada en el punto anterior 2.3.4.

Este apartado resulta bastante extenso y por lo tanto es necesario dividirlo en dos partes. En la primera se estudia la clasificación de las geometrías y en la segunda, los tipos de geometría que se pueden tener.

La clasificación de las geometrías se divide en dos tipos: la geometría **visual**, que se corresponde con la representación visual de un cuerpo, es decir, como es ese cuerpo, su forma, y la geometría de **colisión**, que sirve para poder estudiar las colisiones entre objetos. Estas dos representaciones no tienen por qué coincidir y están definidas con los distintos tipos de geometrías.

- **OpenRave**
  OpenRave define la geometría por medio de la etiqueta **`Geom`**, seguido del atributo **`type`**, que indica el tipo de geometría que se desea representar. La visual y la de colisión se definen utilizando las etiquetas **`data`** y **`render`** respectivamente, como se mostraba en la Figura 2.15. Dentro de la geometría también se establecen las características de posición de la misma.

- **Gazebo**
  En el caso de Gazebo cada clasificación tiene su propia etiqueta.
  Los elementos que representan la interfaz gráfica se indican por medio de la etiqueta **`visual`** y aquellos utilizados para las colisiones por medio de la etiqueta **`collision`**.

  En ambos casos el tipo de geometría se define por medio de etiqueta **`geometry`** y las características de la misma se especifican dentro de esta etiqueta.

- **Comparación de código**

OpenRave

```
<Geom type="trimesh">
  <!--Posición de la malla-->
  <data>./ivs/machosn.iv</data>
  <render>./ivs/machosn.iv</render>
</Geom>
```

Gazebo

```
<collision name="collision1">
  <geometry>
    <!--Definición de la geometría colision-->
  </geometry>
</collision>
<visual name="visual1">
  <geometry>
    <!--Definición de la geometría visual-->
  </geometry>
</visual>
```

Figura 2.19: Comparación de código de geometría visual y de colisión

Los tipos de geometría son la manera de definir la forma del objeto. Puede ser por medio de figuras simples como cubos o esferas o por medio de mallas en 3D.

- **OpenRave**
  Como se ha indicado más arriba la forma geométrica se define dentro del atributo **`type`** y existen cinco tipos:

  - *Caja*
    Se define por medio del atributo **`box`** y sus dimensiones se indican con la etiqueta **`Extents`**, como las semilongitudes de las aristas expresadas en metros.
  - *Cilindro*
    Este atributo se determina como **`cylinder`** y sus dimensiones utilizando las etiquetas **`radius`** y **`height`**, también en metros.
  - *Esfera*
    Se utiliza el nombre **`sphere`** y queda definido con su radio en metros, por medio de la etiqueta **`radius`**.
  - *Plano*
    La forma de definir un plano es por medio del atributo **`plane`** y de la etiqueta **`eqn`**, que determina los parámetros a, b, c y d de la ecuación del plano ax+by+cz+d=0. Con esta ecuación queda definida también la posición de traslación y rotación del plano, por lo que no se precisan otras etiquetas.
  - *Malla*
    Se representan con el atributo **`trimesh`** y el archivo que la define se incluye en el fichero como se indicó en la subsección 2.3.3, en la Figura 2.15.

- **Gazebo**
  Como ya se ha mencionado, el tipo de geometría en Gazebo se define por medio de la etiqueta **`geometry`** y también cuenta con cinco tipos:

  - *Caja*
    Se utiliza la etiqueta **`box`** para definirla y para las dimensiones se utiliza la

etiqueta **size**, en la que se introducen los valores de los lados correspondientes al eje x, y, z en metros.

- *Cilindro*
  La etiqueta para esta geometría coincide con la del atributo en OpenRave, siendo **cylinder** y, de igual manera las características se definen utilizando las etiquetas **radius** y **length**.
- *Esfera*
  La esfera se define utilizando la etiqueta **sphere** y el valor de su radio con la etiqueta **radius**.
- *Plano*
  La etiqueta que lo delimita es **plane** y sus características se definen por medio de otras dos: **normal**, que indica el vector normal de plano y **size**, que indica las dimensiones en la perpendicular al vector normal
- *Malla*
  Se utiliza la etiqueta **mesh** para determinarla y la forma de incluir el archivo que la define se explicó en la subsección 2.3.3 y se puede observar en la Figura 2.16.

- **Comparación de código**

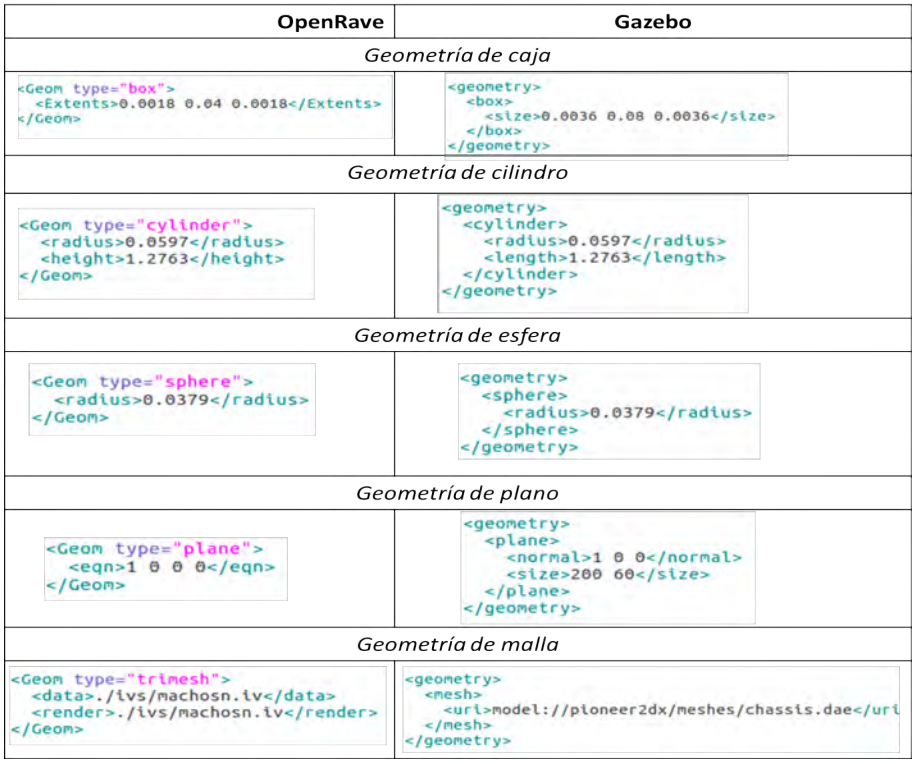

Figura 2.20: Comparación de código de los tipos de geometría

Traslación y rotación

La traslación y la rotación son dos características que es necesario establecer a la hora de definir un cuerpo, sobre todo, si este está formado por varios elementos. Representan la posición del cuerpo en relación al entorno.

- **OpenRave**
  En OpenRave se utilizan dos etiquetas diferentes para cada una de la características.
  - *Traslación:* Para la traslación se utiliza la etiqueta **`traslation`** en la que se indica en metros la posición en el eje x, y, z.
  - *Rotación:* La rotación se define por medio de la etiqueta **`rotationaxis`**, en la cual se indica el eje alrededor del cual se realza el giro estableciendo su valor en "1" y dejando los otros dos como "0". El cuarto dígito señala la magnitud del giro en grados.
- **Gazebo**
  En Gazebo ambas características son definidas con una única etiqueta, sin embargo es una etiqueta diferente para un archivo .sdf y uno .urdf, por lo que se van a explicar ambos.
  - *SDF:* Utiliza la etiqueta **`pose`** seguida del atributo **`frame`**. Dentro de la etiqueta se escriben seis posiciones: las tres primeras representan la traslación en metros y las tres últimas la rotación alrededor de los ejes en radianes.
  - *URDF:* En este caso la etiqueta que lo define es **`origin`** y dentro de la misma tenemos el atributo **`rpy`**, en el cual se indican los tres valores del giro al rededor de los ejes, según la regla de la mano derecha y en radianes; y el atributo **`xyz`**, en el que se describe la posición en cada uno de los ejes, en metros.
- **Comparación de código**

Figura 2.21: Comparación de código de traslación y rotación

Propiedades dinámicas

Dentro de las etiquetas es necesario definir las propiedades dinámicas de los cuerpos, en estos casos se define la masa, la inercia y la posición del centro de gravedad.

- **OpenRave**
  Dentro de OpenRave se define la etiqueta **`mass`** para el grupo de estas propiedades, seguido del atributo **`type`**, que indica el tipo de masa. Estos tipos pueden ser **`box`**, **`sphere`**, **`mimicgeom`** y **`custom`**. Los tres primeros solo necesitan la etiqueta **`total`**, que indica el valor de la masa en kilogramos o **`density`**, que representa la densidad, la inercia y el centro de gravedad se calculan de forma automática con cualquiera de los dos valores anteriores. En el caso de `sphere` también se añade la etiqueta **`radius`**, para indicar el radio.
  Para el tipo **`custom`** esos valores se tienen que definir de forma manual. Las etiquetas **`total`** y **`density`** indican lo mismo que en las masas anteriores, la etiqueta **`inertia`** representa los nueve valores de la matriz de inercia y la etiqueta **`com`** indica la posición del centro de gravedad. Esta masa no se puede aplicar a mallas definidas en 3D ni a planos.

- **Gazebo**
  La etiqueta que define las propiedades dinámicas del cuerpo en Gazebo es **`inertial`** y dentro de la misma se definen las características de forma distinta según el tipo de archivo.

  1. *SDF:* Las propiedades dinámicas se representan con las etiquetas **`mass`** para definir la masa, **`pose`** para el centro de gravedad e **`inertia`** para la matriz de inercia, dentro de la cual se coloca cada término en una etiqueta que indica cual es (por ejemplo **`ixx`** o **`iyz`**).
  2. *URDF:* La etiqueta **`mass`** también define la masa, pero seguida del atributo **`value`**, donde se indica el valor. La posición del centro de gravedad se define con **`origin`**, igual que se indica en el punto anterior, e **`inertia`** representa la matriz, solo que en este caso los valores de cada elemento se colocan dentro del atributo correspondiente (**`iyy`** o **`ixz`**).

- **Comparación de código**

OpenRave

```
<mass type="custom">
  <total>2</total>
  <com>0.35 0 0.02</com>
  <inertia>0.648 0 0 0 0.497 0 0 0 0.0159</inertia>
</mass>
```

Gazebo(SDF)

```
<inertial>
  <mass>2</mass>
  <pose>0.35 0 0.02 0 0 0</pose>
  <inertia>
    <ixx>0.348</ixx>
    <ixy>0</ixy>
    <iyy>0.497</iyy>
    <ixz>0</ixz>
    <iyz>0</iyz>
    <izz>0.0159</izz>
  </inertia>
</inertial>
```

Gazebo(URDF)

```
<inertial>
  <mass value="2"/>
  <origin rpy="0  0  0" xyz="0.35  0  0.02"/>
  <inertia ixx="0.348" ixy="0" ixz="0"
           iyy="0.497" iyz="0" izz="0.0159"/>
</inertial>
```

Figura 2.22: Comparación de código de propiedades dinámicas

Articulaciones o *Joints*

Como ya se ha mencionado, una articulación es la unión entre dos cuerpos. Su función es la de permitir y limitar el movimiento entre ambos.

Para definir una articulación hay que indicar el tipo, los dos cuerpos que la conforman y cual es el padre y cual el hijo, además de características propias de la articulación como son los límites de giro, deplazamiento y velocidad, la posición y dirección de los ejes de giro, etcétera.

- **OpenRave**
  La etiqueta que define una articulación es **`Joint`**, seguida del atributo **`name`**, que define su nombre y el atributo **`type`**, que indica el tipo. Existen cinco tipos de articulaciones: **hinge**, **slider**, **universal**, **hinge2** y **spherical**, que se corresponden con las articulaciones cilíndrica, primática, universal, de doble deslizamiento y esférica respectivamente.

  El resto de características se definen con las siguientes etiquetas dentro de la etiqueta `Joint`:

  - *Etiqueta `body`:* Usada para definir a los dos cuerpos que forman la articulación.
  - *Etiqueta `offsetfrom`:* Indica el cuerpo hijo, es decir, aquel con respecto al cual se realiza el giro.
  - *Etiqueta `anchor`:* Indica la posición del punto de giro.
  - *Etiqueta `axis`:* Señala el eje de giro
  - *Etiquetas `lostop` y `histop`:* Definen el límite inferior y superior respectivamente.
  - *Etiquetas `maxvel` y `maxtorque`:* Establece el valor máximo de velocidad y momento de la articulación respectivamente.

- **Gazebo**
  En este caso la etiqueta principal que define la articulación es igual que en OpenRave, es decir, se utiliza la etiqueta **`joint`**, seguida de los atributos **`name`** y **`type`**. Existen siete tipos de articulaciones: **revolute**, que se corresponde con la de bisagra, **revolute2**, equivale a dos bisagras en serie; **ball**, correspondiente a la esférica; **prismatic** al deslizador; **universal**, es equivalente a una esférica con una grado de libertad restringido; **fixed**, tipo de articulación fija; y por último, **piston**, que se corresponde con un deslizador que puede girar alrededor de un eje.

  Las demás etiquetas se definen dentro de la etiqueta `joint` y describen las características de la articulación:

  - *Etiquetas `parent` y `child`:* Indican respectivamente el cuerpo padre e hijo que conforman la articulación.

- *Etiqueta `pose`:* Establece el punto respecto al que se realiza el giro con el cuerpo hijo como referencia.
- *Etiqueta `axis`:* Establece el eje de giro.
- *Etiqueta `limit`:* Es definida dentro de la etiqueta `axis`. Dentro de esta etiqueta se definen el límite superior e inferior por medio de las etiquetas **`upper`** y **`lower`** respectivamente; la velocidad con la etiqueta **`velocity`**; y **`effort`**, que sirve para indicar el momento que soporta la articulación.

  Hay que recalcar que en el .urdf las etiquetas son las mismas que en el .sdf, pero con alguna sutil diferencia que se detalla a continuación:

  - La etiqueta principal de la definición es la misma, es decir, se usa la etiqueta **`joint`** y los atributos **`name`** y **`type`**.
  - En .urdf dentro de las etiquetas **`parent`** y **`child`** se define el atributo **`link`** en el que se establece el cuerpo correspondiente.
  - La etiqueta **`pose`** del .sdf es sustituida por la **`origin`**, que se ha explicado anteriormente.
  - En .urdf dentro de la etiqueta **`axis`** el atributo **`xyz`** sustituye a la etiqueta del .sdf.
  - Las etiquetas dentro de la etiqueta **`limit`**, son sustituidas en el .urdf por los atributos **`effort`**, **`lower`**, **`upper`** y **`velocity`**.

- **Comparación de código**

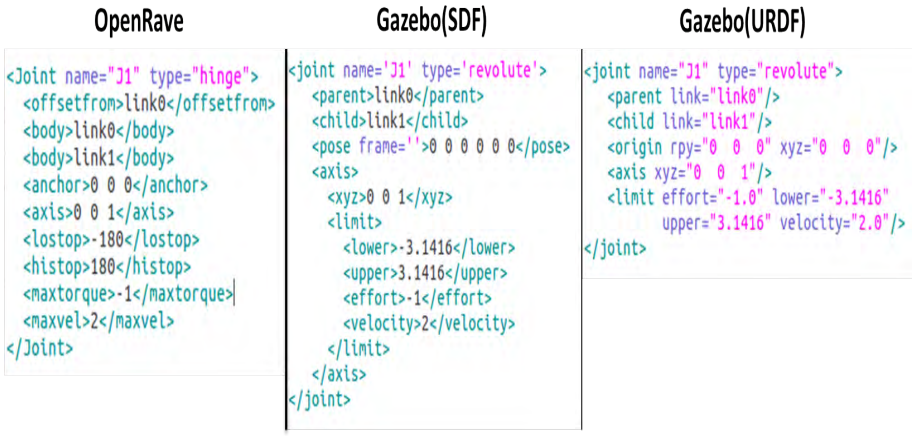

Figura 2.23: Comparación de código de articulaciones o *joints*

Para explicar de forma clara y compacta la gran cantidad de etiquetas presentadas en esta sección se muestra la Tabla 2.1 que indica las diferentes correspondencias. Se utiliza el mismo código de color que en la Figura 2.23 para hacerla más comprensible.

OpenRave	**Gazebo (SDF)**	**Gazebo (URDF)**
Joint	joint	joint
atributos	**atributos**	**atributos**
name	name	name
type hinge hinge2 slider spherical universal	type revolute revolute2 prismatic ball universal fixed piston	type revolute revolute2 prismatic ball universal fixed piston
elementos	**elementos**	**elementos**
body	parent	parent link
body offserfrom	child	child link
anchor	pose	origin
axis	axis xyz limit lower upper effort velocity	axis xyz
lostop histop maxtorque maxvel		limit lower upper effort velocity

Tabla 2.1: Correspondencias entre etiquetas y atributos en articulaciones

La información consultada a lo largo de esta subsección 2.3.4 se puede encontrar en [23], [26], [29] y [41].

Capítulo 3

Arte del modelado

Como se explica en el capítulo 1 el desarrollo de este Trabajo de Fin de Grado queda dividio en dos partes, la primera, la transformación de los archivos que definen a ASIBOT y su entorno al nuevo modelo de simulación (Gazebo), para lo cual es necesario entender los archivos de los que se parte y como se hace la transformación de un modelo a otro, lo cual ha sido analizado y explicado en el capítulo 2 y será aplicado al modelo de ASIBOT en este. La segunda es la creación de trayectorias por medio de RViz y el asistente de MoveIt, parte que será tratada en el capítulo 4.

3.1. Archivos originales

En el capítulo 1 se explica que ASIBOT fue desarrollado en 2006, por lo tanto los archivos para su simulación fueron redactados en el entorno de OpenRave. Sin embargo se está quedando obsoleto y ofrece menos prestaciones frente a Gazebo. Como se ha comentado en el capítulo 2, este modelo utiliza archivos .xml para la descripción del robot y su entorno.
Estos archivos en formato .xml son el punto de partida para este Trabajo de Fin de Grado. En nuestro caso, aunque ASIBOT es un robot doméstico que pretende ser utilizado en todos los puntos de la casa, es decir, cocina, baño, dormitorio, etcétera, se redujeron los puntos de simulación al entorno de la cocina y el baño. Puesto que estos están descritos en un solo archivo con una estructura muy similar, la creación de cualquier otro entorno resultaría sencilla de desarrollar conociendo como funcionan estos archivos, y por lo tanto trivial. Como ejemplo de esto cabe destacar que el archivo del baño no estaba definido en .xml, sino que fue posterior y directamente creado en .sdf.
Los archivos .xml fueron suministrados por los ayudantes de investigación del Robotics Lab y se mostrarán fragmentos de código, aquellos que se consideren apropiados, a lo largo del capítulo para explicar la transformación de los mismos. Se adjuntarán los archivos completos en los apéndices finales.

3.2. Archivos finales

El entorno utilizado actualmente en robótica para la simulación es Gazebo. Este modelo utiliza archivos .sdf para la definión tanto del robot como del ambiente donde este opera. Sin embargo, Rviz y MoveIt utilizan el modelo en .urdf. Aunque son formatos muy similares, puesto que el .sdf es la mejora y actualización del .urdf, tienen las suficientes diferencias como para que la transformación de uno a otro no sea trivial y no permita la reutilización de fragmentos de código.
Se explicará primero cómo se realizó la transformación de .xml a .sdf, para ser capaces de visualizar ASIBOT y los entornos de la cocina y el baño en Gazebo y posteriormente, la transformación de .sdf a .urdf para poder generar las trayectorias.
Al igual que con los archivos originales, se mostrarán los fragméntos de código que se consideren útiles para la explicación de la transformación, y los archivos completos se añadirán al final de la memoria en los apéndices para su consulta.

3.3. Transformación de .xml a .sdf

Puesto que en el capítulo 2 ya se han explicado las características principales de los formatos utilizados por cada simulador y la correlación entre los mismos, en este se aplica esta información para explicar como se hace la transformación entre los modelos.

El punto de partida son cuatro archivos en versión .xml, uno para describir al robot ASIBOT y los otros tres para definir el entorno de la cocina y los elementos integrados en ella. Puesto que las transformaciones de los archivos son muy similares de unos a otros, se ha elegido explicar en detalle la del archivo que define el brazo robótico, puesto que es la que se considera más completa, además de ser el elemento clave en este proyecto.

En primer lugar, en la Figura 3.1 se muestra un esquema de los eslabones (*links*) y articulaciones (*joints*) de los que se compone ASIBOT, al igual que sus sentidos de giro.

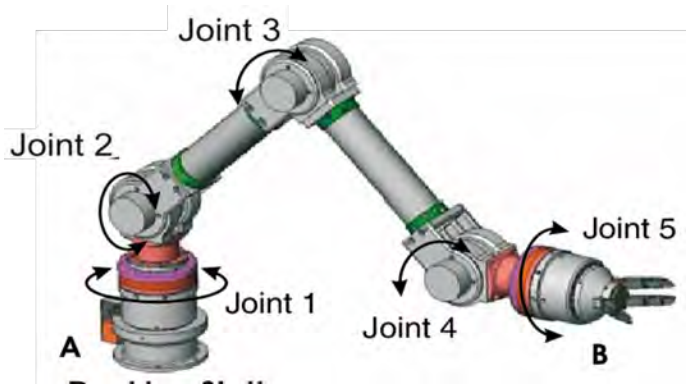

Figura 3.1: Esquema ASIBOT

En total se compone de seis links y sus cinco respectivos joints. Estos son los elementos que hay que definir en el archivo .sdf para conseguir que se visualice en el simulador Gazebo.

A continuación se va a analizar y comparar por secciones ambos códigos explicando también las dificultades encontradas y posteriormente las soluciones obtenidas para ellas.

3.3.1. Encabezado del archivo

En el caso del formato .xml el encabezado del archivo consta solo del nombre del robot definido con la etiqueta `<Kinbody/>`, mientras que en el formato .sdf es necesario definir tanto la versión de .xml como de .sdf utilizada y posteriormente se define el nombre del robot. La Figura 3.2 muestra las diferencias entre estos dos encabezados para el robot ASIBOT.

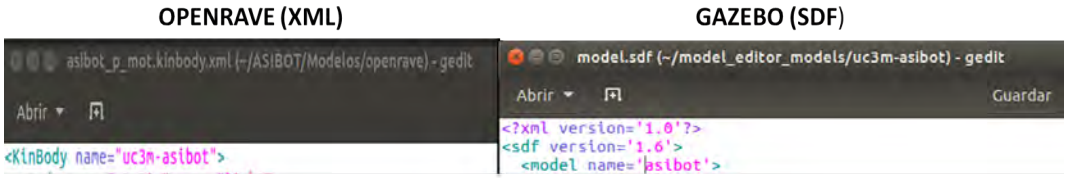

Figura 3.2: Encabezado del archivo .xml y .sdf

Cabe destacar que el nombre del robot en la versión .sdf se modificó de "uc3m-asibot" a solo "asibot", ya que cuando se transformó, posteriormente al .urdf y se aplicó el asistente de MoveIt este daba un error debido al símbolo de "-". Este error supuso una vuelta atrás desde lo que ya se había creado y supuso la modificación de los archivos .sdf para volver a transformalos en .urdf a continuación.

3.3.2. Eslabones

La Figura 3.3 ilustra las diferencias a la hora de definir un eslabón del robot en ambos formatos, para su simplicidad se eligió uno con una única malla.

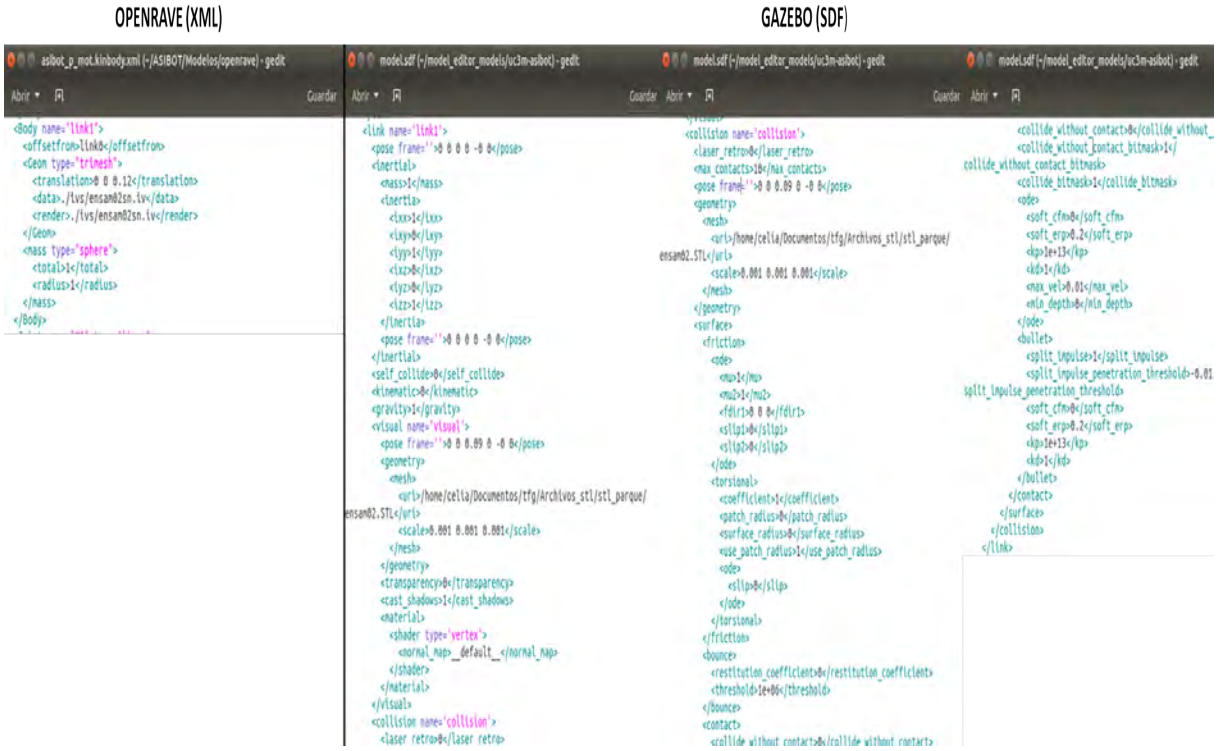

Figura 3.3: Comparación de la definición de un eslabón en el archivo .xml y .sdf

Se puede ver que son muchas las diferencia entre ellos, por lo tanto, para hacer más comprensible las equivalencias entre ambos sistemas se va a utilizar la Tabla 3.1 que se muestra a continuación.

OpenRave (.xml)	**Gazebo (.sdf)**	**Comentarios**
<Body>	<link>	Etiqueta utilizada para definir cada eslabón o elemento del objeto.
<mass>	<inertial>	Estas etiquetas definen la inercia del cuerpo. Mientras que .xml usa la masa y el radio .sdf utiliza el momento de inercia en los ejes.
<total>	<mass>	Etiquetas utilizadas para definir la masa del cuerpo dentro de la etiqueta de inercia.
<render>	<visual>	Define la geometría del objeto.
<data>	<collision>	Define la colisión del objeto.

Tabla 3.1: Diferencias en la definición de un elemento en .xml y .sdf

Como se puede observar en la Figura 3.3 el código .sdf es mucho más largo y extenso que el .xml. Esto se debe a que cuando el archivo es cargado en Gazebo este añade información complementaria sobre el entorno y características adicionales del objeto, como pueden ser el color o material de los mismos o el coeficiente de rozamiento y cuya información no requiere descripción en el formato aplicado en OpenRave.

En el caso de todos los códigos utilizados en este TFG la geometría elegida para trabajar es la de mallas, es decir, cada elemento es definido en un archivo en 3D y posteriormente incorporado al código. En el caso del formato .xml este archivo es un .iv, mientras que en el del .sdf y .urdf es un .stl. Puesto que las mallas en versión .iv también fueron proporcionadas por el Robotics Lab, fue necesario realizar la transformación de un tipo a otro y ya que se trató de un procedimiento complicado, que llevo mucha investigación, se detallará en la subsección 3.3.4.

3.3.3. Articulaciones

Como ya se ha explicado la unión entre dos eslabones (*links*) se denomina articulación (*joint*) y en el caso de ASIBOT son cinco las definidas. Puesto que se considera más clara, se ha decidido aplicar la misma metodología que en la subsección anterior. La Figura 3.4 muestra las diferencias de código entre los dos formatos y la Tabla 3.2 explica las correlaciones entre cada elemento de ambos códigos.

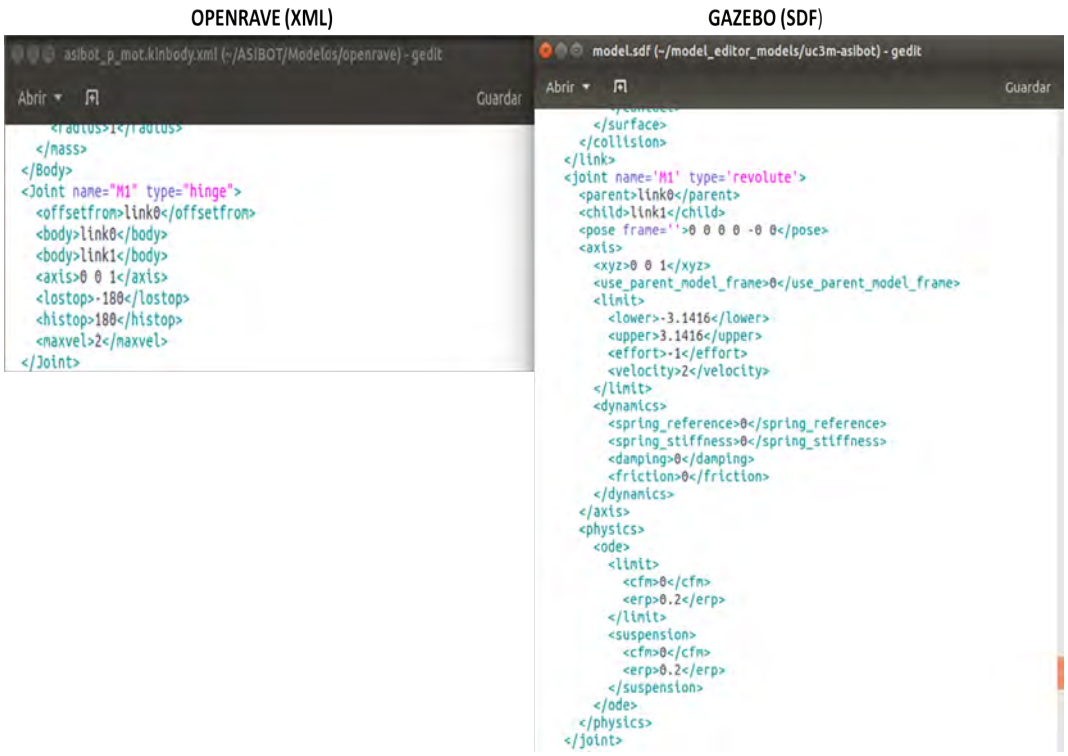

Figura 3.4: Comparación de la definición de una articulación en el archivo .xml y .sdf

OpenRave (.xml)	Gazebo (.sdf)	Comentarios
`type = "hinge"`	`type = 'revolute'`	Define el tipo de articulación.
`<body>`	`<parent>` y `<child>`	La etiqueta `body` en .xml equivale tanto al `parent` como al `child`, es necesario otro elemento para diferenciarlos.
`<offsetfrom>`	`<child>`	`offsetfrom` indica el link que actúa como `child`.
`<lostop>`	`<lower>`	Hace referencia al límite máximo inferior de giro de la articulación. En .xml se expresa en grados mientras que en .sdf se usan radianes.
`<histop>`	`<upper>`	Indica el límite máximo superior.
`<maxvel>`	`<velocity>`	Establece la máxima velocidad de giro de la articulación.

Tabla 3.2: Diferencias en la definición de una articulación en .xml y .sdf

Al igual que ocurre en la subsección de los eslabones, el modelo en .sdf cuenta con una mayor cantidad de información que el .xml. De nuevo se trata información complementaria añadida por el simulador al código, una vez que el archivo es cargado dentro del mismo.

3.3.4. Transformación de mallas de .iv a .stl

Como se ha indicado en la subsección 3.3.2 el formato de mallas .iv no era válido para ninguna de la versiones utilizadas en este TFG (.sdf o .urdf), por lo que fue necesario realizar una conversión de uno a otro. Sin embargo, este proceso no fue tan sencillo como podía parecer puesto que el formato .iv resulta bastante antiguo. A continuación se presenta una lista con los diferentes problemas que se abordaron en esta conversión y al final de la misma se detalla la solución a la que se llegó.

1. En primer lugar, recalcar que se investigaron y buscaron múltiples programas para intentar realizar la conversión. Sin embargo, varios de ellos o bien no disponían de una versión gratuita lo suficientemente potente como para hacerla o bien eran programas pago, y puesto que este se trata de un proyecto individual no era posible comprar un programa cuya única función era solo transformar unos pocos archivos.

2. El segundo de los problemas a los que se hizo frente fue el hecho de que varios programas podían importar un archivo .iv, pero no eran capaces de exportarlo a .stl, o bien esta transformación se hacía de forma incorrecta y el archivo obtenido era completamente inútil. *FreeCad* (solo guardaba en formato .FCStd), la versión de prueba de *OpenInventor* (no permitía exportar a ningún formato) y *3D Transform* (el archivo exportado en stl era incorrecto) son algunos ejemplos de programas utilizados que daban estos problemas. Dentro de este punto se podría añadir el hecho de que muchas veces aunque se tenía la opción de enviar el archivo a .stl no era posible cargar el .iv, como sucedía con el programa *Solid Edge*.

3. Finalmente se consiguió realizar la transformación utilizando el programa *Polytrans*. Sin embargo la versión gratuita que se pudo descargar eliminaba un tanto porciento de la malla, concretamente cada quinto polígono [42], haciendo que esta quedase con agujeros, y planteando un nuevo problema a abordar: el relleno de dichas mallas para presentar una buena representación de las piezas.
   Cabe destacar que con ayuda de uno de los ayudantes de investigación se intentó obtener la versión completa de *Polytrans* para poder obtener toda la malla en la transformación, pero fue imposible.

En esta segunda lista se presentan los problemas afrontados en la búsqueda para encontrar la forma de completar los agujeros de las mallas.

1. En primer lugar, varios de los programas que se investigaron tenían problemas abriendo el archivo .stl o bien requerían demasiada potencia como para que el equipo utilizado (un ordenador ASUS) pudiera soportarla y se quedaban bloqueados sin posibilidad de operar con ellos. Como ejemplo del primer caso está *Sketchup* y en el caso del segundo se tiene *Meshlab*.

2. El programa *Meshmixer* parecía capaz de dar una solución a este problema. Sin embargo, aunque se itentaron tres métodos diferentes no fue posible alcanzarla:

   - La primera opción que se utilizó no fue adecuada, en primer lugar porque servía para cerrar agujeros que tuviera la pieza en sí y en segundo, porque resultaba casi imposible seleccionar el interior de los agujeros dejados por la falta de malla, como indicaba el vídeo [43].
   - La segunda opción consistía en seguir, una vez importado el archivo, la siguiente cadena de acciones: *Analysis->Inspector->* selección del tipo, por ejemplo *flat fill-> Auto repair all*. No obstante, las piezas resultaban demasiado complicadas y el programa no era capaz de detectar de forma correcta los huecos y el relleno resultaba erróneo [44].

- El último método fue el más complejo y el más cercano a ofrecer una solución. Este método consistía en, una vez importado (*Import*) y seleccionado el archivo deseado, se podía desplegar un menú (*Edit->Make solid*) que permitía jugar con el tamaño, densidad y número de los triángulos de la malla, utilizando los comandos *Offset distance*, *Solid accuracy* y *Mesh density*. Sin embargo, aunque en la algunas de las piezas se consiguió mejorar la malla, estas quedaban más infladas y algo deformadas, por lo tanto no se pudo aceptar como solución válida [45].

3. Finalmente, por medio del programa *Geomagic Freeform* se consiguió rellenar los agujeros aplicando la opción "Llenar orificio", que completaba las piezas antes de permitir trabajar con ellas.

Se querría destacar el hecho de que el tipo de formato .stl al que se realizaba la transformación es importante, puesto que el válido en Rviz es el binario. Sin embargo los programas no siempre hacen una disticción entre las diferentes opciones, por lo tanto una vez realizada la transformación era necesario cerciorarse de que el formato obtenido era un .stl binario.

A continuación y a modo de resumen, se va a explicar todo el procedimiento final paso a paso, que fue seguido para obtener los nuevos archivos. Los tres primeros pasos se corresponden con la trasformación por medio del programa *Polytrans* de .iv a .stl y los tres siguientes a la forma de rellenar los huecos con el programa *Geomagic Freeform*:

1. Una vez iniciado el programa es necesario importar el archivo .iv, para ello se selecciona la opción de "Import 3D Geometry" y se elige la opción "Inventor(SGI) .iv 2.0 file", lo cual lleva a elegir el archivo de la carpeta donde esté almacenado. Esta secuencia se puede ver en la Figura 3.5.

2. Una vez que el archivo es seleccionado, es cargado y el programa le aplica la reducción en la malla, por lo tanto ya se pueden observar los agujeros que posteriormente será necesario completar. Esto se puede apreciar en la Figura 3.6.

3. Cuando el programa termina de cargar, se exporta a .stl por medio de la opción "Export 3D Geometry" y se selecciona el formato "STL file" como se puede observar en la Figura 3.7. Una vez hecho se selecciona la carpeta y el nombre del archivo, quedando realizada la transformación, obteniendo un archivo en formato stl, pero con las mallas incompletas.

4. Una vez que se obtiene el modelo en .stl se abre el programa *Freeform* y se carga el archivo por medio del comando "Importar modelo", que lleva directamente a elegir el archivo deseado. Este proceso se puede observar en la Figura 3.8.

5. El siguiente paso es seleccionar el tipo de relleno ("Llenar estilo") que se desea en la barra de herramientas de la parte inferior, como se muestra en la Figura 3.9. En este caso se selecciona la opción "Llenar orificio" y, se elige "Aplicar".

6. El paso anterior completa la malla y cierra los agujeros de la misma. Para guardar el archivo se escoge la opción "Exportar modelo", y al guardarlo, es necesario tener en cuenta que el formato seleccionado es "Binary STL file". La Figura 3.10 ilustra este paso a la vez que muestra la pieza rellenada.

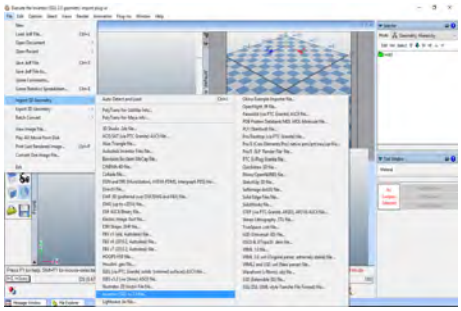

Figura 3.5: Importación del archivo .iv

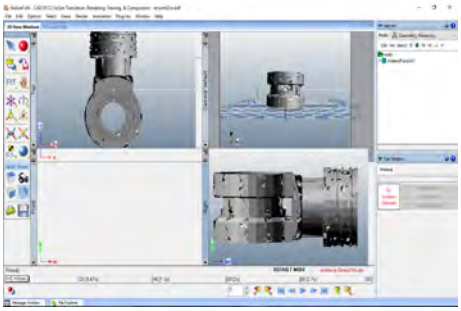

Figura 3.6: Carga del fichero .iv

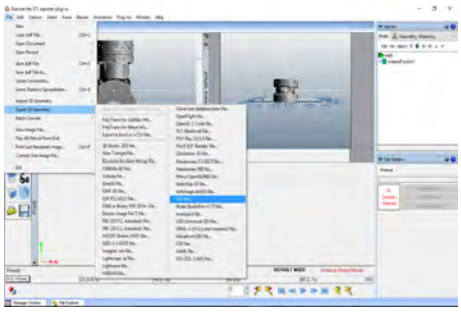

Figura 3.7: Exportación al formato .stl

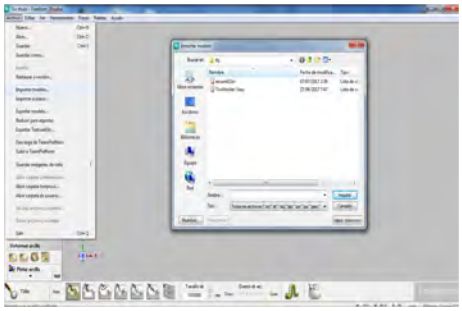

Figura 3.8: Carga del archivo .stl de *Polytrans*

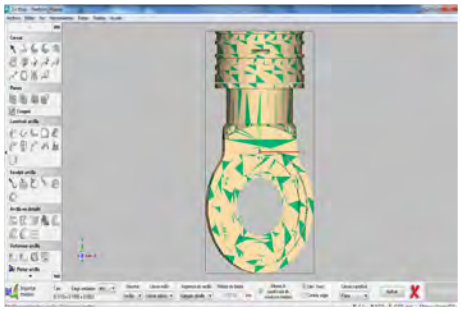

Figura 3.9: Relleno de los huecos

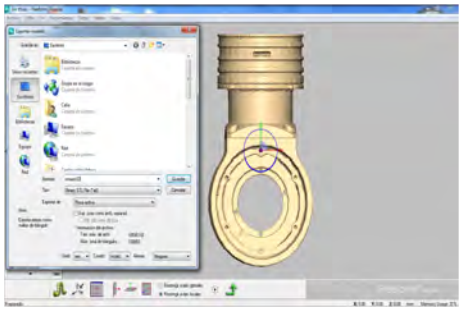

Figura 3.10: Exportación a formato .stl binario

Como se puede apreciar, la transformación de estos archivos aunque parecía trivial en un principio luego resultó no serlo, y supuso muchas horas de investigación que finalmente dieron su fruto y permitieron obtener unos resultados bastantes buenos. La Figura 3.11 muestra la malla del ensam02 de ASIBOT tras la transformación con *Polytrans* y su paso por *Freeform*, pudiéndose observar una mejora considerable.

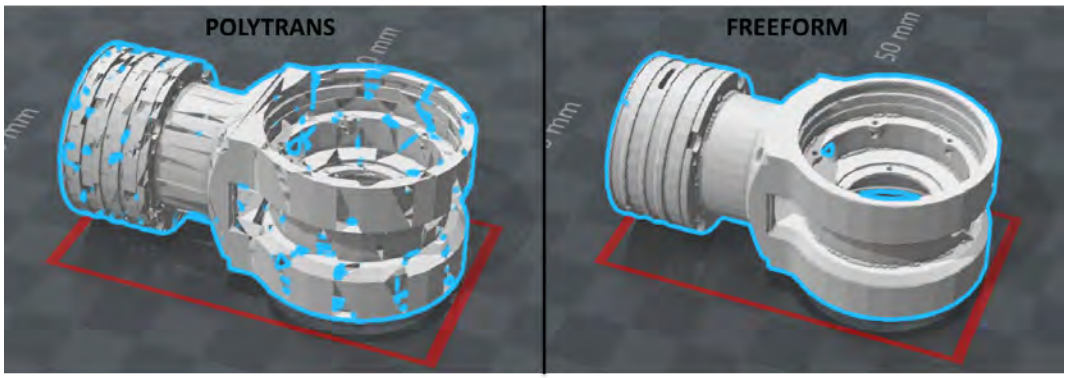

Figura 3.11: Comparación entre los archivos de *Polytrans* y *Freeform*

Este procedimiento se siguió con todos los elementos de ASIBOT al igual que con los del entorno, es decir, la cocina, la silla de ruedas con la persona y la base que sirve de anclaje al robot. Sin embargo, aunque funcionó bien con la mayoría de los archivos, en la cocina no fue posible rellenar los huecos de forma precisa, lo cual le da ese aspeco incompleto que se podrá apreciar en las demostraciones. Una posible forma de arreglar ese archivo es volver a crearlo desde cero y en formato .stl utilizando un programa de modelado 3D y sustituir la nueva malla en el código. Sin embargo, puesto que la cantidad de tiempo en este TFG era limitada se deja como futuras mejoras del mismo.

3.3.5. Archivos del entorno

Como ya se ha mencionado en varias ocasiones, los entornos en el que se ha realizado esta simulación con ASIBOT son la cocina y el baño. Por lo tanto ha sido necesario trabajar con cuatro archivos más: la cocina, la silla de ruedas (incluída a la persona sentada en ella), sistema de anclaje que sirve como base al robot y el baño.
Para los tres primeros se siguió el mismo procedimiento que se ha explicado a lo largo del capítulo, es decir, del archivo antiguo de OpenRave se obtuvo el nuevo modelo para Gazebo, aplicando las correlaciones entre las diferentes etiquetas explicadas en las Tabla 3.1 y Tabla 3.2 y posteriormente cargándolos en el simulador para que este terminase de completar los códigos. Por lo tanto el código y la comparación entre ellos no se va a exponer a continuación, sino que serán añadidos en los anexos de la memoria.

El archivo del baño fue creado directamente desde cero y en formato .sdf, puesto que no se contaba con un original en .xml. Pero puesto que ASIBOT tiene un carácter doméstico se quiso desarrollar este entorno y crear una trayectoria para el mismo, a pesar de no estar definido inicialmente.
Ya que la forma de modelar su archivo .sdf es extremadamente parecida al entorno de la cocina, se utilizó este de modelo. Por lo que era necesario obtener una malla 3D que lo definiese. Para ello se buscó en una página de archivos de CAD llamada *GRABCAD COMMUNITY* [46]. Una vez obtenido el archivo .stl que modelaba un baño en 3D se realizó un corte por medio del comando “Dividir” (dentro de “Editar”) del programa *3D*

Builder, el objetivo de esta operación era conseguir una vista abierta del mismo, en la que resultase más sencillo apreciar los movimientos del robot. La Figura 3.12 ilustra el baño con y sin el corte.

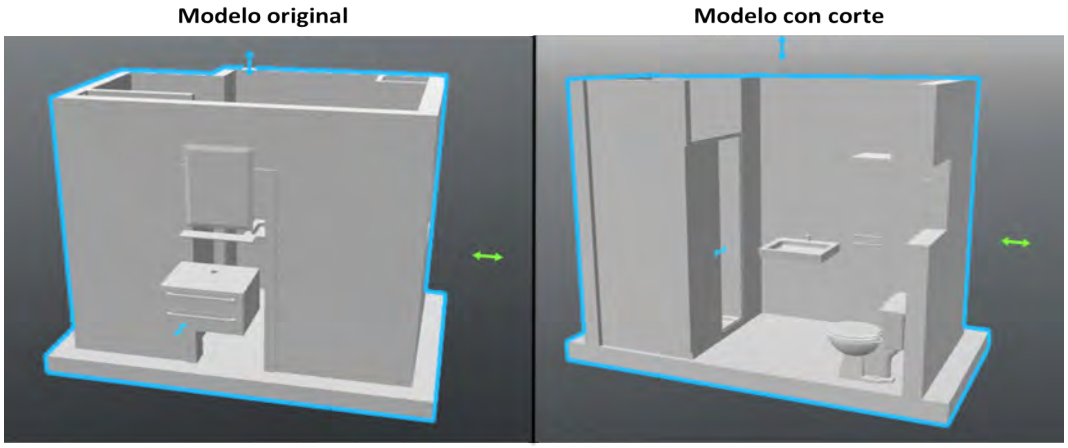

Figura 3.12: Modelo en 3D original del baño y una vez realizado el corte

Para crear el archivo .sdf se tomó el de la cocina y se sustituyó la malla por la del baño y por último se introdujo en gazebo de forma que el simulador terminase de completar el código, al igual que con el resto de elementos.

3.3.6. Resultados

Puesto que esta primera parte del TFG ya proporcionaba unos resultados que se pueden apreciar por sí solos, estos son presentados a continuación.
Por un lado las Figura 3.13 y 3.14 presentan el entorno de la cocina con la silla de ruedas y ASIBOT desde un ángulo lateral y superior respectivamente, y la Figura 3.15 muestra el entorno del baño también con ASIBOT y la persona.

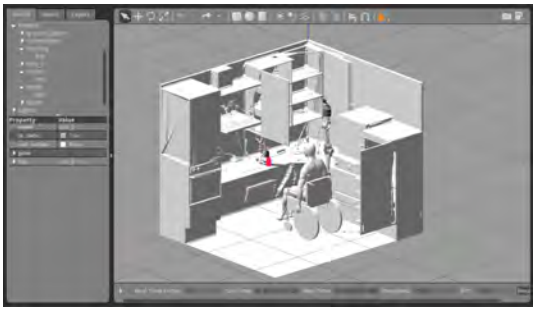

Figura 3.13: ASIBOT en el entorno de la cocina completo

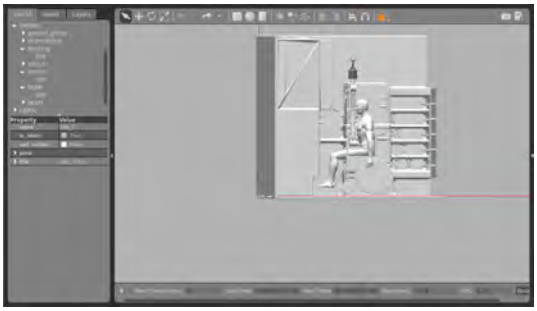

Figura 3.14: ASIBOT en el entorno de la cocina, vista lateral

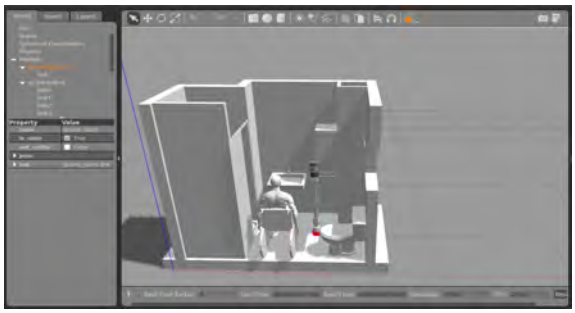

Figura 3.15: ASIBOT en el entorno del baño

Capítulo 4

Creación de Trayectorias

En este capítulo se va a desarrollar y explicar la segunda parte de este Trabajo de Fin de Grado, es decir, la creación de trayectorias por medio del asistente de MoveIt y Rviz, y como objetivo adicional, la posterior conexión con Gazebo para poder visualizarlas en el entorno de simulación.
Para poder completar esta tarea hubo que reducirla a tres subtareas que era necesario superar en orden:

1. Transformar los .sdf en .urdf, pues es el formato utilizado por MoveIt y Rviz.
2. Utilizar el asistente de MoveIt para generar los archivos necesarios para Rviz y generar las trayectorias dentro del mismo.
3. Conectar Gazebo y Rviz para exportar las trayectorias creadas en Rviz al simulador.

Para poder explicar con claridad cada una de ellas se va a dividir el capítulo en tres secciones donde se desarrollen cada una de estas tareas y las dificultades encontradas en cada caso.

4.1. Transformación de archivos .sdf a .urdf

Como ya se ha explicado en los capítulos 2 y 3 los modelos del .sdf y .urdf son muy parecidos, pues el .sdf deriva del .urdf, pero las diferencias entre ellos son lo suficientemente significativas como para no poder aprovechar fragmentos completos de código.
Puesto que la transformación a mano y de nuevo de los cinco archivos que componen ASIBOT y su entorno resultaba demasiado tediosa y larga, se decidió investigar formas automáticas para ello. Tras consultar varias fuentes se encontró una forma de realizar la transformación entre ambos archivos por medio del paquete *pysdf*.

4.1.1. Paquete *pysdf*

Paquete utilizado para analizar el .sdf en una estructura jerárquica y transformarlos a .urdf [47].
Para poder utilizarlo era necesario descargarlo y compilarlo como un paquete de ROS, para hacer eso se siguieron varios pasos:

1. Selección de la carpeta donde se desea instalar el paquete, para ello se utiliza el comando `cd` de ROS [48], en concreto se eligió la carpeta "src", dentro del espacio "catkin_ws", quedando el comando final como: "`$ cd /home/celia/catkin_ws /src`".
2. Descarga del paquete por medio del comando "`$ git clone https://gith-ub.com/andre-asBihlmaier/pysdf.git`" [49], que descarga el paquete desde la dirección web señalada.
3. Cuando el paquete está descargado es necesario compilarlo, para ello se sitúa la terminal en la carpeta "catkin_ws" utilizando la orden `cd ..` [48] y se compila usando el comando `catkin_make` [50].

4.1.2. Conversión de los archivos

Una vez que el paquete está instalado la transformación resulta bastante sencilla y se realiza por medio de un solo comando en la terminal, pero antes de eso hay que concretar en que carpeta se quiere generar el nuevo archivo. Esto se realiza de nuevo por medio del comando `cd`. En este caso se decidió que fuera en el Escritorio, por lo tanto el comando completo sería: "`$ cd /home/celia/Escritorio`". Tan pronto como se está dentro de la carpeta deseada se escribe el siguiente comando: "`$ rosrun pysdf sdf2urdf.py model.sdf model.urdf`".

- **rosrun**: Comando de gazebo que permite correr un nodo dentro de un paquete, usando directamente el nombre del paquete sin necesidad de conocer la dirección al mismo [51].
- ***pysdf***: Paquete que queremos utilizar.
- **sdf2urdf**: Nodo dentro del paquete *pysdf* que se utiliza en la conversión.
- **model.sdf**: Archivo en .sdf que se quiere transformar, debe encontrarse en la carpeta especificada, en este caso el Escritorio.
- **model.urdf**: Nombre con el que el archivo .urdf es generado, como ya se ha dicho se crea en el espacio que se haya definido anteriormente con el comando `cd`.

[52] Sin embargo, la transformación realizada por este paquete no es perfecta y es necesario efectuar una modificación en la definición de las mallas tridimensionales:

- En el archivo generado automáticamente la dirección donde se encuentra el archivo stl que define la malla viene dada por "`package://PATHTOMESHES/Documentos/tfg/Archivos_stl/girados/Modelos_girados/docking_s.stl`". La definición de "`pa-ckage://PATHTO-MESHES`" resulta errónea y provocaba que el archivo no pudiera encontrar el fichero, generando un error que se muestra en la Figura 4.1.

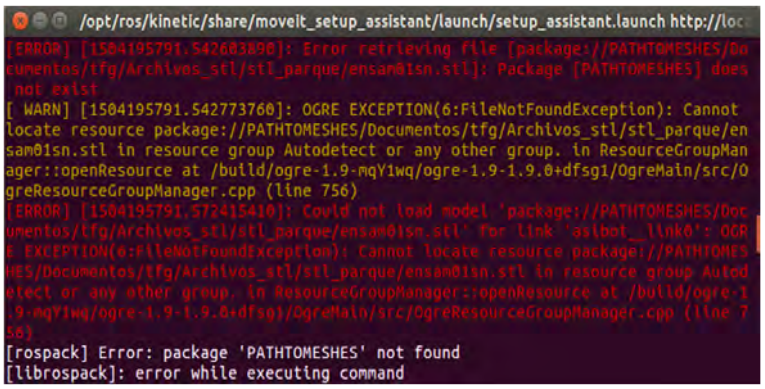

Figura 4.1: Error al cargar el fichero

- Existen dos formas de redefinir la dirección para que sea correcta: "`file://home/home/celia/Documentos/tfg/Archivos_stl/girados/Modelos_girados/docking_s.stl`" o bien "`file:///home/celia/Documentos/tfg/Archivos_stl/girados/Modelos_girados/do-cking_s.stl`". Es necesario tanto añadir dos veces el comando `/home` o utilizar la tercera "/", porque si no el primer argumento tras las dos barras es considerado el nombre de usuario y, por lo tanto, se carga una dirección errónea y impidiéndose de nuevo el acceso a los archivos [53].

Para ilustrar mejor la diferecia en las definiciones de esta dirección hasta el archivo, se presenta la Figura 4.2

Figura 4.2: Definición de la dirección al fichero

Una vez que se tiene el archivo .urdf correctamente definido, lo cual puede ser comprobado por medio del comando: "`$ check_urdf asibot.urdf`" [54], se puede empezar a trabajar con el asistente de MoveIt.

4.2. Asistente de MoveIt y planificación de trayectorias con Rviz

4.2.1. MoveIt

MoveIt es un software de última generación para la manipulación móvil, que incorpora los últimos avances en planificación de trayectorias, manipulación, percepción 3D, cinemática, navegación y control. Pone al servicio del usuario una plataforma sencilla de usar para desarrollar aplicaciones robóticas, evaluar los diseños de diferentes robots y crear productos robóticos integrados para distintas aplicaciones. Además, MoveIt cuenta con una larga lista de robots, que pone a disposición de todos sus usuarios, aunque si no se dispone del modelo que se desea en ella, también permite trabajar con prototipos de creación propia, que es lo que se desarrolla en este TFG.

Instalación de MoveIt

El primer paso para el uso del programa es la instalación del mismo, para ello se ejecutan dos comandos a través de la terminal de Ubuntu:

- "`$ sudo apt-get install ros-kinetic-MoveIt`" : Instala los paquetes necesarios para el programa. Es necesario especificar la versión de ROS con la que se trabaja, en este caso *Kinetic*.
- "`$ source /opt/ros/kinetic/setup.bash`": Configura el entorno de trabajo.

Una vez que todo está instalado y configurado se puede empezar a utilizar el asistente de MoveIt [55].

Asistente de MoveIt (*Setup Assistant*)

El asistente es el siguiente paso en el uso de MoveIt. Se trata de una interfaz gráfica que permiete al usuario configurar cualquier robot para su uso en MoveIt, y posteriormente en Rviz. Su función principal es la de crear un archivo SRDF (*Semantic Robot Description Format*, archivo donde se guardan las características definidas en el asistente) pero además, también genera otros archivos de configuración necesarios para la comunicación entre los elementos de MoveIt, que posteriormente también son necesarios para la conexión con Gazebo.
La creación de estos archivos se hace en un total de ocho pasos:

1. **Lanzamiento y carga del archivo**: Para lanzarlo se utiliza el comando: "`$ roslaunch MoveIt_setup_assistant setup_assistant.launch`" que abre la ventana que se observa en la Figura 4.3. Desde ahí se selecciona "*Create New MoveIt Configuration Package*", lo que lleva a elegir el archivo en el que está descrito el robot con el que se quiere trabajar. Hay que tener en cuenta que este archivo tiene que ser en formato .urdf, motivo por el cual era necesario realizar la transformación explicada en la sección 4.1.2. Si el archivo estuviese definido en un xacro la conversión se realizaría por medio del comando: "`$ rosrun xacro xacro.py model_xacro > model.urdf''`, como se indicó en la subsección 2.3.2, antes de introducirlo en el asistente. Una vez seleccionado el archivo se carga por medio del botón "*Load Files*". Este paso también queda reflejado en la pantalla mostrada en la Figura 4.3.

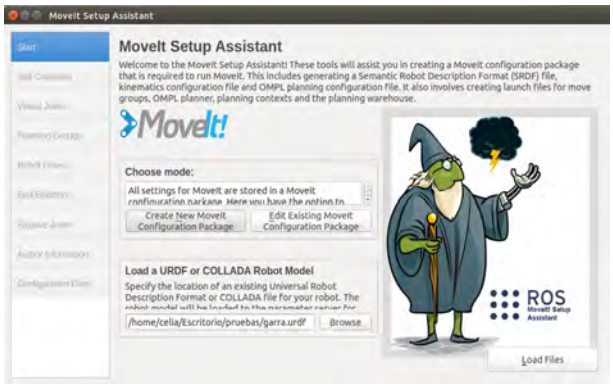

Figura 4.3: Lanzamiento del asistente de MoveIt y carga del archivo

2. **Generación de la matriz de colisiones**: Dentro de la pestaña "*Self-Collisions*", la opción "*Regenerate Default Collision Matrix*" identifica de forma automática que eslabones puede ser deshabilitados del control de colisión. De esta forma se reduce el tiempo de procesamiento. En la Figura 4.4 se puede observar la categorización de los eslabones.

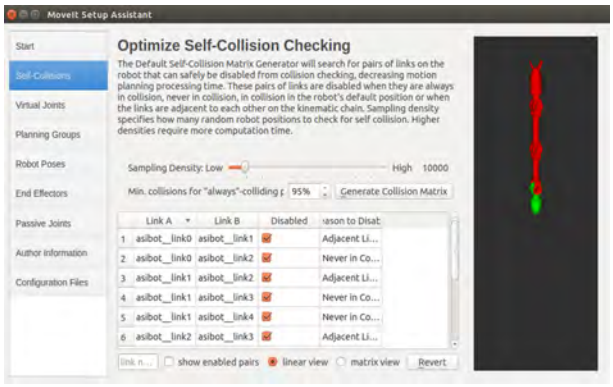

Figura 4.4: Generación de la matriz de colisiones

La etiqueta que aparece en la parte superior como "Sampling Density" indica el número aleatorio de posiciones a analizar para definir la colisión, si el número es muy elevado el coste computacional será elevado, pero la colisión será menor. En este caso se dejó el valor por defecto de 10.000.

3. **Articulaciones virtuales**: La siguiente pestaña es la de "*Virtual Joints*", que sirven para anclar el robot al mundo donde se trabaja. No es un elemento que sea obligatorio añadir, de hecho en aquellos que son estáticos y no se mueven no es necesario. Sin embargo, sí que lo es en aquellos casos donde la posición de la base es móvil y por lo tanto es necesario crear una articulación virtual con respecto al marco *odom* (*odom frame*).

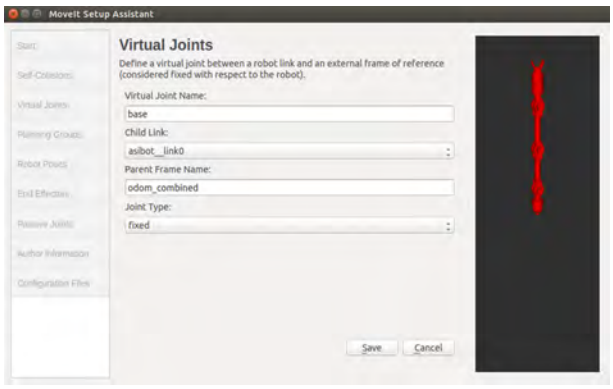

Figura 4.5: Definición de articulaciones virtuales

En el caso de este proyecto se definió el "asibot_link0" como articulación virtual, con las características que se observan en la Figura 4.5.

4. **Definir los grupos para la planificación**: Estos grupos no son más que un conjunto de eslabones y articulaciones que se mueven de forma conjunta para alcanzar la posición final o la herramienta del extremo del brazo.
   Para la creación de un grupo es necesario seleccionar el botón "*Add Group*", que lleva al menú mostrado en la Figura 4.6, donde se pueden seleccionar distintos parámetros para el grupo, las articulaciones y eslabones que se desean añadir. Esta selección se hace a través de otro menú, ilustrado en la Figura 4.7, donde se eligen los elementos que se desean añadir, se confirman por medio de la flecha a la derecha y por último se guarda la elección en el botón "*Save*". En la Figura 4.8 se presenta otra forma diferente de definir el grupo, por medio de una cadena cinemática.
   En el caso de ASIBOT se crearon dos grupos:

   - **Cuerpo**: Denominado como *arm*, se definió como una cadena cinemática formada por los eslabones desde el "asibot__link0" al "asibot__link5", eligiendo como *kinematic solver* el *plugin* KDL.
   - **Pinza**: Utilizado para denominar al extremo final de ASIBOT, tanto a la garra como a la herramienta que se coloque según la trayectoria deseada. Designado con el nombre de *pinza*, es definido por medio del "asibot__link6" y "asibot__M6" en este trabajo.

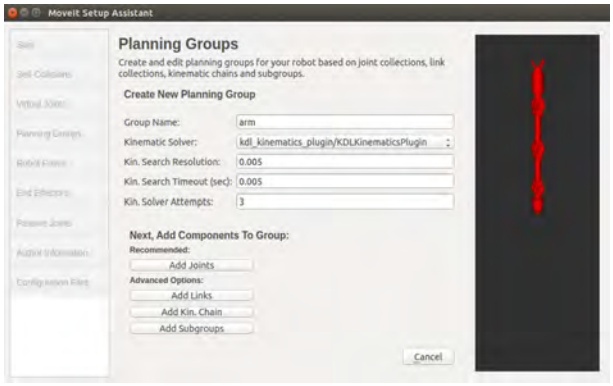

Figura 4.6: Parámetros del grupo

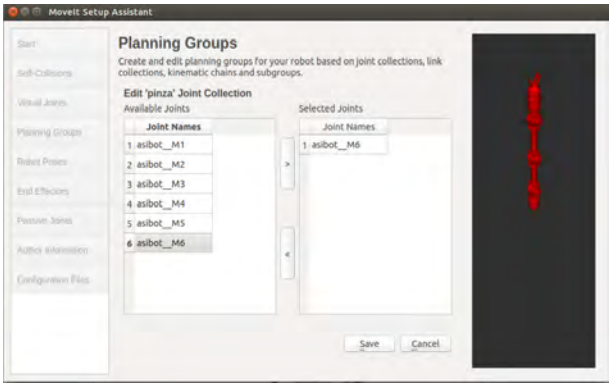

Figura 4.7: Elección de articulación del grupo pinza

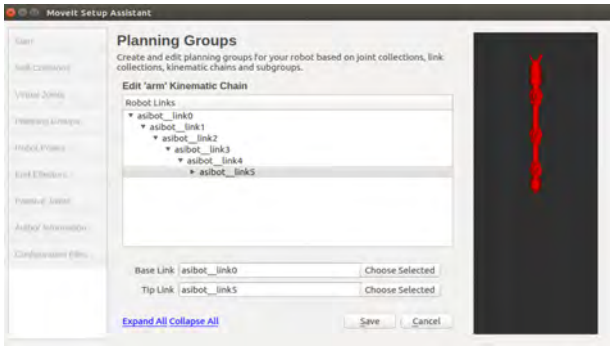

Figura 4.8: Cadena cinemática para el grupo *arm*

MoveIt cuenta con una infraestructura de *plugins* que permite al usuario desarrollar su propios algoritmos cinemáticos inversos. El *plugin* por defecto para MoveIt es el "*KDL numerical jacobian-based solver*", que es configurado automáticamente por el asistente [56].
EL KDL (Kinematics and Dynamics Library) desarrolla un espacio de trabajo independiente para el modelaje y computación de cadenas cinemáticas (como robots, figuras animadas por ordenador, etcétera), cadenas cinemáticas de varias familias (como por ejemplo humanoides) y sus especificaciones de movimiento e interpolación [57].

Cuando todos los grupos han sido creados y guardados se puede apreciar una pantalla como la de la Figura 4.9.

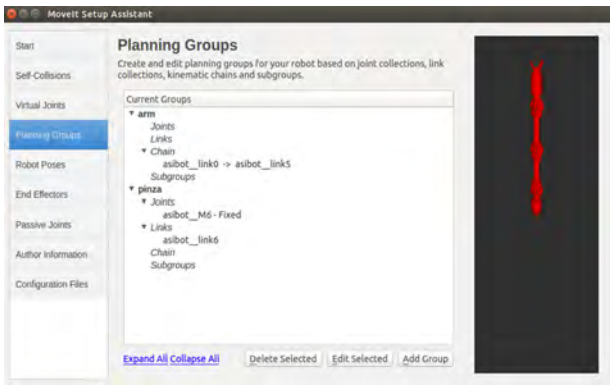

Figura 4.9: Lista de los grupos

5. **Posición del robot**: En la pestaña "*Robot Poses*" es posible definir distintas posiciones para el robot, que luego dentro de Rviz pueden ser seleccionadas y utilizadas para realizar las trayectorias. La Figura 4.10 muestra el menú de esta pestaña. Es necesario darle un nombre a la posición y elegir el grupo sobre el que se desea operar. El movimiento se realiza seleccinando cada articulación. Es un buen punto para comprobar que las definiciones son correctas y que los límites de cada articulación están bien definidos.

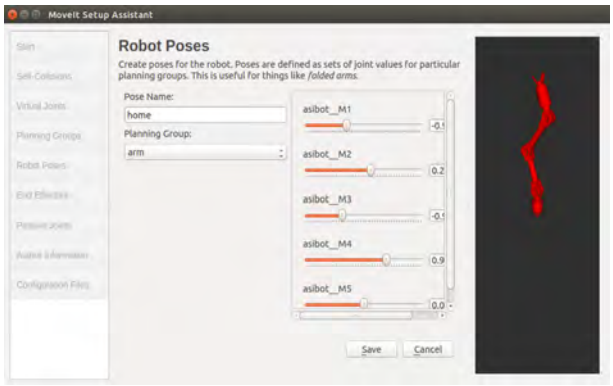

Figura 4.10: Definición de una posición para el robot

6. **Definición de herramientas extremas**: La siguiente pestaña es la de "*End Effectors*", en la cual se establece el nombre del robot y la articulación, a la vez que se elige el grupo que la define, el eslabón padre (*parent link*) y el grupo padre (*parent group*). En la Figura 4.11 se pueden ver las selecciones que se hicieron para ASIBOT, aunque en este punto se pueden definir tantos elementos como se quieran. En el caso de este proyeto solo era necesario definir uno para la garra o la herramienta que se fuese a utilizar.

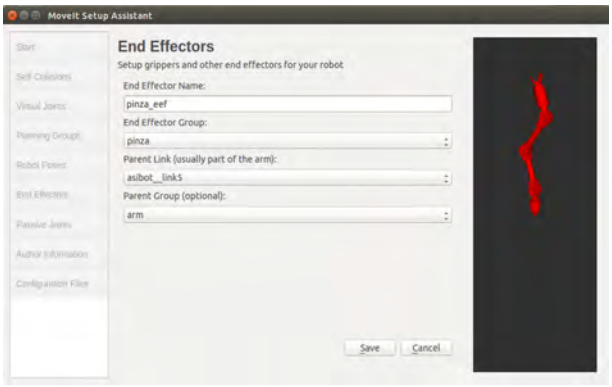

Figura 4.11: Definición de herramientas extremas

7. **Añadir articulaciones pasivas**: Dentro de la pestaña "*Passive Joints*" se especifican las articulaciones pasivas del robot, es decir, aquellas que no tienen actuadores. A la hora de la creación de la trayectoria el simulador ignorará este tipo de articulaciones. En el caso de ASIBOT no se tiene ninguna.

8. **Generación de los archivos de configuración**: El paso final consiste en la creación de los archivos. Esto se realiza dentro de la pestaña "*Configuration Files*". Es necesario seleccionar la carpeta donde se quieren generan, en este caso se decidió crear una carpeta en la dirección "home/celia/catkin_ws/src" denominada "asibot". Una vez elegida la carpeta, el botón "*Generate Package*" produce los diferentes archivos y se puede cerrar el asistente de MoveIt (botón "*Exit Setup Assistant*"). Este paso final queda reflejado en la Figura 4.12.

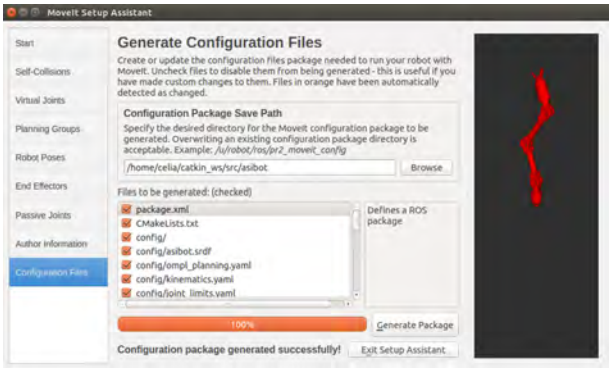

Figura 4.12: Generación de los archivos de configuración

Tras la creación de los archivos por medio de MoveIt se pueden empezar a planear las trayectorias utilizando el programa Rviz [58].

4.2.2. Rviz

Rviz es un visualizador 3D asociado a ROS que permite mostrar información sobre los sensores y los estados desde ROS. Además, utilizando Rviz se puede visualizar la configuración actual del robot en un modelo virtual, a la vez que se pueden mostrar representaciones a tiempo real sobre los valores de los sensores. Sirve, como se mencionó en el capítulo 1, para obtener todos los datos posibles sobre el modelo real, pero utilizando el virtual [59].

Todo lo necesario para comenzar la generación de trayectorias se obtiene en el punto anterior, por lo tanto lo único que queda es lanzar el programa por medio del comando "`$ roslaunch asibot demo.launch`", que está compuesto de tres partes:

1. **roslaunch**: Paquete dentro de ROS. Es la forma estándar de inicializar los nodos de ROS y el robot que se quiere utilizar [60].
2. **asibot**: Nombre de la carpeta donde se encuentran los archivos generados con MoveIt, es el paquete creado por el asistente.
3. **demo.launch**: Archivo .launch que se encuentra en la dirección "./catkin_ws/src/asibot/launch", permite explorar todas las características de este paquete.

Este comando abre el programa Rviz con el modelo de ASIBOT que se había definido en el .urdf. Sin embargo, es necesario realizar una serie de modificaciones en los parámetros del programa antes de poder empezar a diseñar las trayectorias:

- **Añadir el *Motion Planning Plugin***: En la pestaña "*Display*" se selecciona el botón "*Add*", se elige la opción "*Motion Planning*" y se guarda pulsando "OK".
- **Definir el *Fixed Frame***: En la subpestaña "*Global Options*" se establece el *"Display Fixed Frame*" como "*/odom_combined*".
- **Configuración de los *plugins* del robot**: Dentro de la subpestaña "*MotionPlanning*":
  - Comprobar que en "*Robot Description*" esté seleccionado "*robot_description*".
  - Establecer "*planning_scene*" dentro del campo "*Planning Scene Topic*".
  - Dentro de "*Planning Request*" cambiar el "*Planning Group*" al grupo con el que queremos crear la trayectoria, en este caso se puede elegir entre "*arm*" o "pinza".

- Por último, seleccionar "*/move_group/display_planned_path*" en "*Trajectory Topic*", que se encuentra dentro de "*Planned Path*".

- **Librería OMPL**: Es necesario comprobar que está cargada, en caso negativo sería imposible generar las trayectorias. Esto es de hecho, uno de los problemas que se afrontó en este TFG como se discutirá en la subsección 4.2.2.

Cuando todos los elementos quedan comprobados y configurados se puede apreciar una imagen en la pantalla como la que muestra la Figura 4.13 [61].

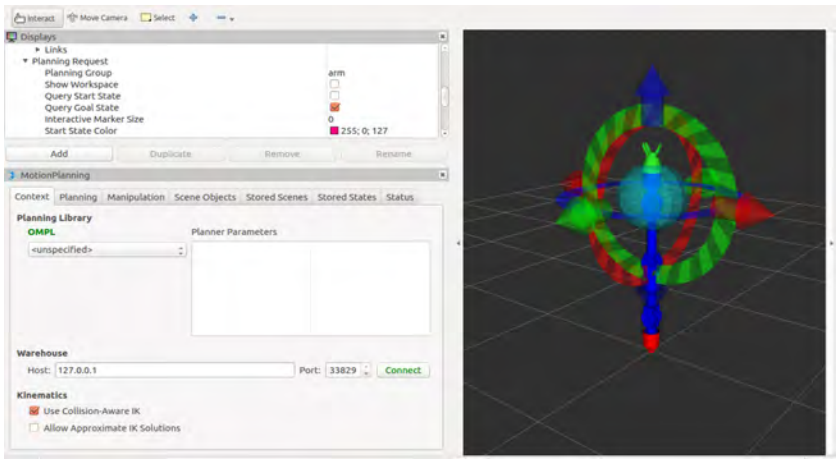

Figura 4.13: Entorno de Rviz para generar trayectorias

Librería OMPL y problemas asociados a ella

En los párrafos anteriores se ha explicado que una de las condiciones para poder crear las trayectorias en el programa Rviz, es que este tuviese cargada la librería OMPL.
Esta librería es una de las múltiples opciones que proporciona Rviz y es, además, la librería que carga por defecto.

OMPL (Open Motion Planning Library) es una librería de planificación de movimiento de software libre utilizada en la creación de trayectorias. MoveIt se integra con la OMPL y utiliza sus planificadores de movimiento como propios y por defecto. Hay que tener en cuenta que los planificadores en OMPL son abstractos, es decir, no tienen el concepto de robot, por lo tanto MoveIt configura la OMPL para poder trabajar con problemas robóticos [56].

Error al cargar la librería: Según lo explicado, la librería debe ser cargada automáticamente por Rviz si los archivos de MoveIt y el .urdf están bien generados, sin embargo en el caso de este TFG las primeras versiones de ASIBOT con las que se trabajó no permitían que se cargase.

Tras el lanzamiento del archivo por medio de la terminal se podía observar no solo que la librería OMPL no era cargada en Rviz, sino que además se obtenía el error que se muestra en la Figura 4.14.

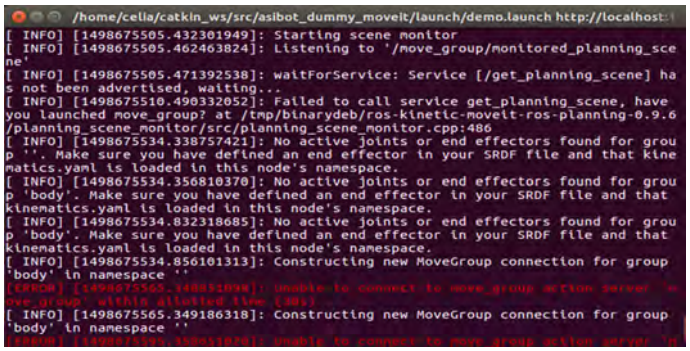

Figura 4.14: Error de conexión al lanzar el archivo

El error que se mostraba por terminal determinaba que se trataba de un problema de conexión, en concreto que no era posible conectarse o lanzar el servicio "/get_planning_scene". En un intento de comprender a qué podía deberse se compararon los archivos de ASIBOT con los del robot PR2 [63] y se observó que practicamente no existía ninguna diferencia entre ambos, sin embargo, sí mostró que el orden de ejecución de las operaciones entre ambos modelos tras el lanzamiento se veía alterado. Como consecuencia de esta observación se estableció la premisa de que MoveIt lanza varios servicios en paralelo, que tienen dependencias entre sí y en el modelo de ASIBOT, a causa del supuesto retardo, el recurso que utilice alguno de esos servicios no ha sido cargardo por completo y por lo tanto impide la conexión.
Ante esta situación se abrieron dos vías de acción:

- **Generación del paquete de MoveIt**
  La primera de estas vías consistía en analizar el proceso de creación de archivos con el asistente, en concreto las articulaciones virtuales, pues generaban algunas advertencias dentro de la terminal, como la que se puede ver en la Figura 4.15.

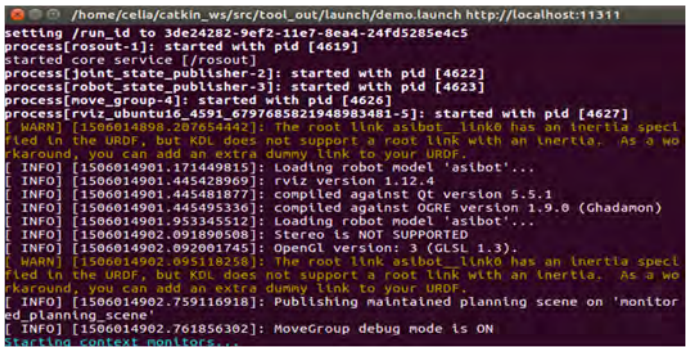

Figura 4.15: Advertencia vinculada a las articulaciones virtuales

Sin embargo, tras definir el robot sin articulaciones virtuales y definirlas en otros eslabones (se llegó a crear el denominado "*dummy link*", que era un eslabón auxiliar), el problema de conexión seguía apareciendo, por lo que se consideró una vía sin salida.

- **Utilización de archivos muy pesados**
  Al lanzar el programa se podía apreciar que tardaba mucho en cargar el modelo de ASIBOT, por lo tanto se consideró la posibilidad de que estos archivos pesasen tanto que realentizaban el proceso y por lo tanto impedían la conexión.
  Para comprobar esta teoría se creó un modelo con las mallas de ASIBOT obtenidas en *Polytrans*, es decir, aquellas que tenían la malla incompleta. Efectivamente, debido al menor peso de estos archivos el programa era capaz de realizar la conexión entre servidores, y por lo tanto de cargar la librería OMPL.

  No obstante esto presentaba un nuevo problema, pues no era posible utilizar los archivos obtenidos en el capítulo 3, ya que eran extremadamente pesados (probablemente porque la forma de rellenar las mallas con *Freeform* no era exacta), pero tampoco se podían utilizar los obtenidos con *Polytrans*, pues estaban tan llenos de agujeros que no permitían apreciar bien los componente de ASIBOT.

  Tras una larga investigación dentro de los repositorios de archivos que existen sobre ASIBOT se consiguió encontrar unos modelos en .stl que tenían un peso mucho menor. Con estas mallas se definió un nuevo archivo .urdf, que se utilizó para crear un nuevo paquete con el asistente de MoveIt, consiguiéndose finalmente cargar la librería OMPL en Rviz y poder empezar a planear trayectorias.

  En realidad, puesto que se decidió crear trayectorias para distintos entornos de la casa, como se explicará en el capítulo 5, se crearon tres archivos diferentes para cada una de ellas, pues utilizaba una herramienta distinta y de esta forma se evitaba la reescritura tanto del archivo .urdf como de aquellos generados por MoveIt.

Definición de nuevos archivos .urdf

Como se ha comentado unas líneas más arriba, se crearon varios archivos .urdf, en concreto tres, uno para la trayectoria de rehabilitación, otra para la de la cocina y un último para el baño. Sin embargo, para el correcto funcionamiento de ASIBOT en Rviz fue necesario modificar levemente el archivo .urdf que se había obtenido al comienzo de este capítulo proveniente del .sdf creado en el capítulo 3.

Para conseguir un funcionamiento adecuado del extremo libre se añadió un eslabón extra (denominado "asibot__link6") donde se definía la herramienta que se utilizaba en cada recorrido; la garra en la trayectoria de rehabilitación, la herramienta del vaso en la de la cocina y por último, la herramienta para sujetar un cepillo de dientes, incluyendo un modelo del mismo [46], para la del cuarto de baño. Por consiguiente también fue necesario añadir una nueva articulación entre el "asibot__link5" (actuando como padre) y el "asibot__link6" (actuando como hijo), que se definió como fija, es decir, sin posibilidad de giro.

Además, en el archivo del baño fue necesario realizar una modificación adicional, pues no era posible cargar el modelo por medio del procedimiento explicado en el siguiente punto (sección 4.2.2), ya que era cargado con una escala varias veces superior a la de ASIBOT y a tanta distancia que no era posible situarlos en el mismo entorno. Para solucionar este problema se decidió definir el baño como un eslabón extra de ASIBOT (se creó el "bano_link"), en el cual se aplicó la reducción de escala correspondiente y después se unió a la base del robot por medio de una articulación fija entre el "asibot__link0" y el "bano_link", que definía también la posición que ocuparía ASIBOT dentro de este entorno.

Tras la creación de un .urdf para cada modelo se les aplicó a los tres el asistente de MoveIt, generando los paquetes necesarios asociados a cada uno de ellos, permitiéndose de esta forma comenzar con la creación de trayectorias, que se explica en el siguiente punto.

Planificación de trayectorias

Una vez que el modelo está cargado en Rviz y los valores que se comentan en la primera parte de esta subsección están ajustados, crear una trayectoria resulta una tarea menos complicada.

- **Introducción de elementos del entorno**
  Según la trayectoria que se desea crear el entorno de trabajo cambia, por lo tanto cuando se lanza Rviz se lanza solo el modelo del robot, en este caso ASIBOT, y después se añade el área de trabajo, en el caso de este TFG la cocina y el baño.
  La introducción de estos elementos es bastante cómoda. En la pestaña "*Scene Objects*" se selecciona la opción de "*Import file*", lo cual lleva a seleccionar el archivo de mallas (formato .stl) que se desea introducir. Una vez seleccionado es posible modificar su posición y escala por medio de los comandos que se muestran en la Figura 4.16.

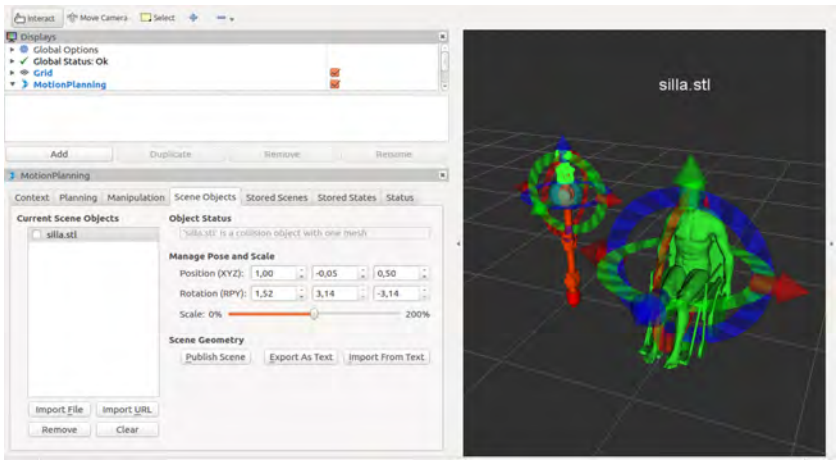

Figura 4.16: Añadir modelos al entorno en Rviz

Puesto que cada vez que se desee crear una trayectoria es necesario cargar y colocar los archivos del entorno, este puede ser guardado como un archivo de extensión **.scene**, por medio del botón "*Export As Text*". De esta forma, en dicho fichero se almacenan los elementos introducidos en ese entorno y sus posiciones, pudiéndose importar todos a la vez por medio del botón "*Import From Text*" la próxima vez que se quiera utilizar ese entorno específico.

- **Establecimiento de la posición inicial y final**
  A la hora de crear la trayectoria, Rviz permite establecer la posición inicial y final entre las que se desea que el robot realice el movimiento.

  - *Posición inicial:* Dentro de la etiqueta "*Planning*" se selecciona "*Select Start State*", que permite elegir la posición inicial que se desea. Cuenta con cuatro opciones: ***random valid*** (posición aleatoria dentro de los límites del espacio de trabajo), ***random*** (posición aleatoria), ***current*** (posición seleccionada por el usuario) y ***same as goal*** (posición igual a la posición final). Si además se ha definido alguna posición dentro del asistente de MoveIt también da opción a seleccionarla.
    En este caso la opción seleccionada es ***current***. Una vez confirmada por medio del botón "*Update*" aparece una nueva visualización del robot que se puede llevar a la posición deseada por medio de las flechas y arcos que aparecen en torno al extremo del mismo. Un ejemplo de posición inicial para ASIBOT se puede apreciar en la Figura 4.17 en rosa.
  - *Posición final:* Dentro de la misma etiqueta "*Planning*" se puede seleccionar "*Select Goal State*", para definir la posición final deseada. Cuenta con las mismas cuatro opciones que la inicial, excepto que en vez de ***same as goal*** la última posición es ***same as start***. También puede seleccionar las posiciones definidas en el asistente de MoveIt.
    La definición de la posición final se realiza igual que la inicial, tras seleccionar la opción ***current***, se establece la posición del robot por medio de los arcos y flechas de su extremo. La Figura 4.17 muestra un ejemplo de posición final para ASIBOT en azul.

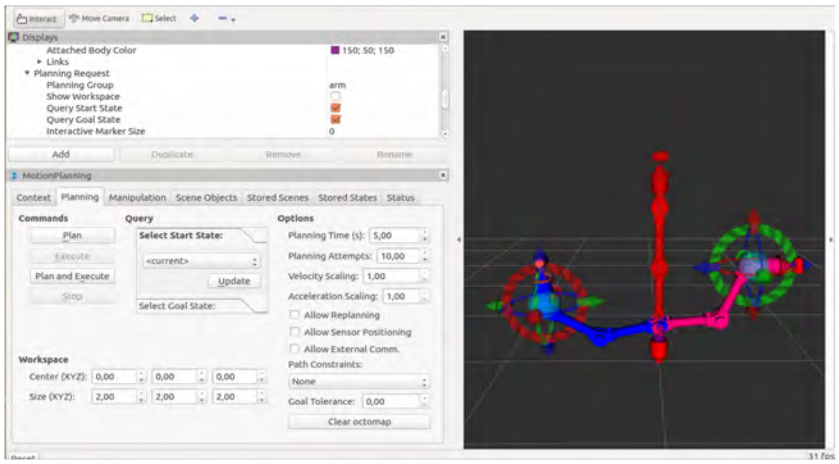

Figura 4.17: Ejemplo de posición inicial y final para ASIBOT

- *Validez de las posiciones:* Puesto que se hace una definición de las colisiones cuando se define el modelo tanto en .sdf y .urdf como al generar sus archivos en MoveIt, Rviz es capaz de indicar cuando dos o más cuerpos entran en colisión, lo que permite crear trayectorias válidas. La forma que tiene de indicarlo es por medio de un cambio de color en la zona en colisión. En las Figuras 4.18 y 4.19 se presentan unas colisiones en el entornos de la cocina y el baño, respectivamente, marcadas en color azul claro.

Figura 4.18: Colisión en el entorno del baño

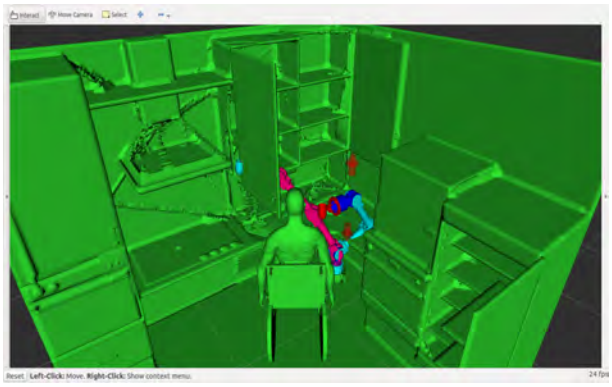

Figura 4.19: Colisión en el entorno de la cocina

Además, el programa permite definir el entorno de trabajo o *workspace*, que limita el alcance máximo del robot. De esta forma, cuando se selecciona una posición fuera de él el programa lanza una advertencia. Para ASIBOT el alcance máximo es de 1300mm [62].

Tan pronto como ambas posiciones y el entorno están definidos, se ejecuta la trayectoria pulsando el botón “Plan”.

Una vez que la trayectoria está creada es posible exportarla a Gazebo, para lo cual

es necesario establecer una conexión entre los dos programas (Gazebo y Rviz), que es el punto estudiado en la sección 4.3.

4.3. Conexión Gazebo y Rviz

Una vez conseguido el objetivo de crear las trayectorias por medio de MoveIt, se quiso implementar la conexión entre Gazebo y Rviz, para de esta forma exportar dichas trayectorias del segundo al primero.

4.3.1. Creación de archivos

En este punto se estudiará como establecer la interfaz entre el paquete de configuración de MoveIt con Gazebo, para de esta forma ser capaz de ejecutar en el segundo las trayectorias creadas con el primero. Para conseguir este objetivo es necesario crear un controlador de trayectorias que tenga la interfaz *FollowJointTrajectoryAction* (interfaz de acción de ROS de la cual se sirve el robot para realizar sus acciones [56]).

La creación de esta interfaz consiste en una serie de pasos que se detallan a continuación y que se siguieron del libro *Mastering ROS for Robotics Programming* [64]. La creación de estos archivos se hace en su carpeta correspondiente, en el caso de este proyecto en el paquete generado por MoveIt en el punto 8 de la subsección 4.2.1 "asibot". Además fue necesario crear otros dos paquetes en la dirección "home/celia/catkin_ws/src" llamados "asibot_gazebo" y "asibot_description", utilizando el comando "`$ catkin_create_pkg asibot_gazebo std_msgs rospy roscpp`" [65], habiéndose utilizado previamente el comando "`cd`" para situarse en la carpeta citada.

1. **Creación del archivo de los controladores:** Se trata de un archivo de configuración para establecer la comunicación entre los controladores de trayectoria en Gazebo y MoveIt. El nombre del archivo es **controllers.yaml** y se tiene que crear dentro de la carpeta "config", que a su vez se encuentra en el paquete generado por MoveIt (carpeta "asibot" que fue mencionada en el punto 8 de la subsección 4.2.1).

   El contenido de este archivo son los elementos que definen los dos grupos de controladores, en el caso de este proyecto el "*arm*" y la "pinza".

2. **Creación del archivo de lanzamiento de los controladores:** La función de este archivo es la carga y lanzamiento de los controladores de trayectorias creados en el punto anterior. Se encuentra dentro de la carpeta "launch" generada en el paquete de MoveIt (carpeta "asibot") y es denominado **asibot_MoveIt_controller_manager.launch**.

3. **Creación del archivo de configuración de los controladores de Gazebo:** Archivo generado dentro de la carpeta "config" del paquete "asibot_gazebo" y denominado **trayectory_control.yaml**. Este fichero contiene la lista de los controladores de ROS que es necesario cargar junto con Gazebo.

4. **Creación del archivo de lanzamiento de los controladores de Gazebo:** Este archivo carga el modelo del robot a la vez que los controladores, creados en el apartado anterior, en Gazebo y, al mismo tiempo, lanza el paquete necesario para inicializar los nodos de MoveIt y abrir RViz. Puesto que es un fichero de lanzamiento se crea dentro de la carpeta "launch" del paquete "asibot_gazebo", con el nombre **asibot_bringup_MoveIt.launch**.

   Sin embargo, aunque según el libro este debería ser el último archivo a crear, lo cual permitiría lanzar ambos programas simultáneamente y comenzar con la exportación de trayectorias, se observó que este último fichero contenía archivos que no habían sido definidos y que por lo tanto era necesario crear. Para ello se buscaron los archivos correspondientes al robot utilizado como modelo en el libro ("seven_dof_arm") y se adaptaron para el modelo de ASIBOT. Estos archivos se obtuvieron de un repositorio de códigos [66], del cual solo se adquirieron aquellos necesarios para la conexión. El proceso utilizado para realizar esa conexión partió del archivo **asibot_bringup_MoveIt.launch** y se observó a qué archivos referenciaba, se fueron construyendo en los paquetes correspondientes y observando a qué archivos llamaban a su vez. Puesto que en total se crearon 5 archivos adicionales se presenta la Tabla 4.1 donde se indica el nombre de todos aquellos necesarios para la conexión y la carpeta correspondiente donde se encuentran.

asibot_gazebo/launch	asibot_gazebo/config	asibot/launch	asibot/config	asibot_description/urdf
asibot_bringup_MoveIt				
asibot.world (1)				asibot.urdf (2)
asibot_gazebo_states (1)	asibot_gazebo_joint_states (2)			
asibot_trayectory_controller (1)	trayectory_control (2)			
		asibot_MoveIt_controller_manager (1)	controllers (2)	
		MoveIt_planning_execution (1)		
		move_group (2)		
		MoveIt_rviz (2)		

Tabla 4.1: Jerarquía de los archivos necesarios para la conexión Rviz-Gazebo

Es necesario aclarar algunos aspectos de la tabla para que pueda ser entendida fácilmente:

- **Código de color:** Se ha utilizado el mismo color para aquellos archivos que están relacionados, además de una secuencia numérica que indica qué archivo llama a cuál. Es decir, el archivo que llama se marca con un (1) y el llamado tiene un (2).
- **Colocación en la tabla:** Para que resulte más sencillo saber qué fichero llama a cuál, además del color y el valor numérico, se han establecido a la misma altura, para aquellos que se encuentren en columnas diferentes, y los que se encuentran dentro de la misma columna y son llamados por otro archivo dentro de esta, se han colocado debajo y con un espacio tabulador.
- **Extensión de los archivos:** Aquellos ficheros que se encuentran dentro de una carpeta "launch" tienen una extensión **.launch** y, los de las carpetas "config" la **.yaml**. Estas extensiones se han omitido para disminuir el tamaño de la tabla.

Una vez creados estos archivos es posible lanzar la conexión por medio del comando "`$ roslaunch asibot_gazebo asibot_bringup_MoveIt.launch`". Este comando llama al archivo dentro de la carpeta "asibot_gazebo", que a su vez lanza los archivos correspondientes mencionados en la Tabla 4.1. Sin embargo, al lanzarlo en este proyecto surgieron una serie de problemas que se detallarán a continuación, junto con las soluciones que se presentaron para los mismos:

1. El primer problema surgía cuando al lanzar el comando, se abría tanto Gazebo (cargando dentro el modelo de ASIBOT) y Rviz (donde no se cargaba dicho modelo), pero lanzaba por terminal la advertencia que se muestra en la Figura 4.20.

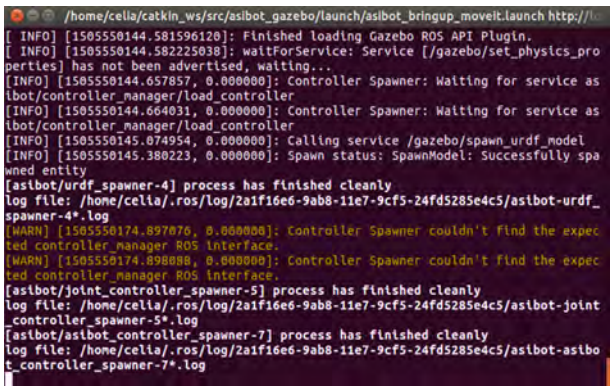

Figura 4.20: Advertencia sobre el *controller_manager*

Tras la investigación de dicha advertencia se encontró que podía deberse a dos factores: que no estuviese en ejecución el *controller_manager*, o bien que este y el *spawner* se encontrasen en un *namespace* diferente.
Ante el primer caso, se averiguó que la causa era la ausencia de un *plugin* de ROS que no se estaba cargando (gazebo_ros_control). Para solucionar ese problema se adjuntó el fragmento de código que se muestra en la Figura 4.21 dentro del archivo "asibot.urdf" que contiene la descripción del robot [67].

```
<gazebo>
  <plugin name="gazebo_ros_control"
filename="libgazebo_ros_control.so">
      <robotNamespace>/asibot</robotNamespace>
  </plugin>
</gazebo>
```

Figura 4.21: Código añadido al .urdf para cargar el plugin gazebo_ros_control

Sin embargo, esto no solucionaba el problema de la conexión.

2. Con respecto al segundo factor mencionado en el punto anterior, al volver a lanzar los archivos se podía observar por la terminal que no siempre se cargaba el *namespace* asociado al grupo de controladores utilizado.

Para solventar este problema se estudió y modificó el archivo **spawner** situado dentro de una de las librerías de ROS, en el que se localizó la línea de código que lanzaba la advertencia de la Figura 4.20 y, se añadió sobre ella una línea de código que establecía el *namespace* de todos los archivos como `asibot/`. A su vez, para evitar problemas de repetición y sobreescritura se eliminó de todos los archivos incluidos en la carpeta "asibot_gazebo/launch" el comando `ns=/asibot`.

3. Cuando el problema de las conexiones entre el *controller_manager* y el *spawner* quedó solucionadas por medio de los dos puntos anteriores, se presentaron nuevos retos, en este caso relacionado con los controladores, como se puede apreciar en la Figura 4.22.

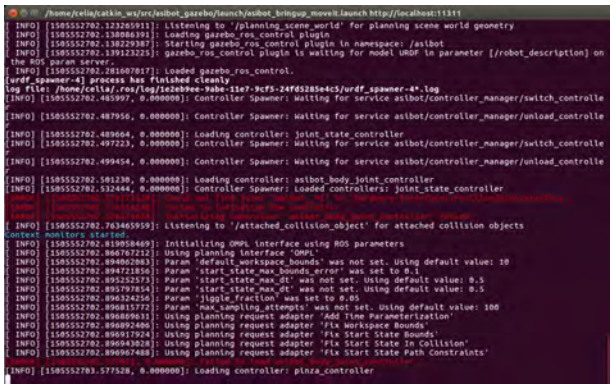

Figura 4.22: Error controladores

Esto es debido a la necesidad de introducir una etiqueta referenciando al *hardware* de la interfaz asociada a los joints y actuadores del robot. Para ello se utilizó la etiqueta **hardwareInterface** como se indica en la Figura 4.23 [68].

```
<transmission name="trans1">
 <type>
  transmission_interface/SimpleTransmission
 </type>
 <actuator name="A1">
 </actuator>
 <joint name="asibot__M1">
    <hardwareInterface>hardware_interface/PositionJointInterface</
hardwareInterface>
 </joint>
</transmission>
```

Figura 4.23: Fragmento del código asociado al *hardware* añadido al .urdf

Sin embargo, al realizar esta modificación se observó que era necesario instalar el paquete de ROS "gazebo-ros-control" por medio del comando: "`$ sudo apt-get install ros-kinetic-gazebo-ros*`" y el paquete de controladores asociado a las articulaciones, con el comando "`$ sudo apt-get install ros-kinetic-joint*`". Ambas órdenes analizan los paquetes que ya están instalados y actualizan e instalan aquellos elementos dentro de los mismos que sean necesarios.
Una vez dispuestos todos los paquetes necesarios se podía observar una salida como la que se muestra en la Figura 4.24, en la que ya no solo se observa que no existen

advertencias ni errores, sino que además se establecen las conexiones necesarias. A pesar de esto el programa se quedaba bloqueado, impidiendo la unión que se buscaba.

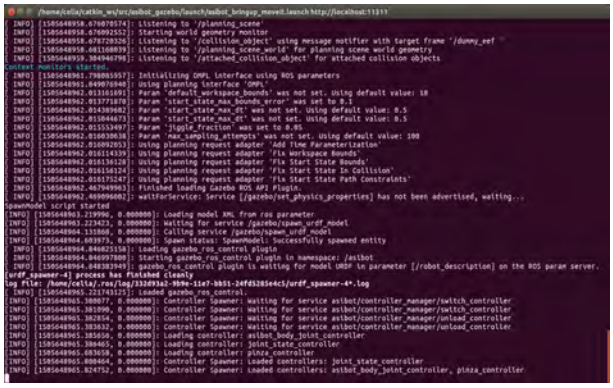

Figura 4.24: Salida por terminal sin avisos

Además, en la Figura 4.25 se puede observar la lista de servicios y controladores que se encuentran en funcionamiento mientras se lanza la conexión, por lo tanto se puede afirmar que sí se está realizando una conexión, pero que por algún motivo esta queda bloqueada.

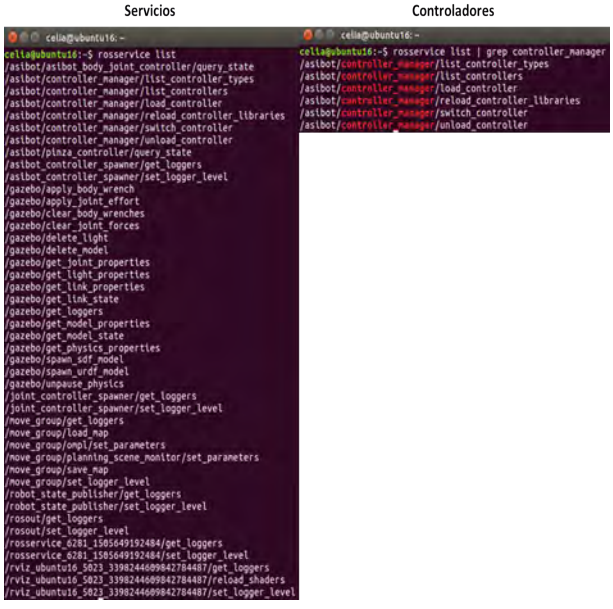

Figura 4.25: Servicios y controladores activos durante la conexión

Puesto que no se contaba con errores o advertencias que pudiesen guiar la investigación en alguna dirección se decidió abrir tres líneas de investigación:

- **Estudio del modelo "seven_dof_arm"**
  La primera vía consistía en lanzar los archivos del robot "seven_dof_arm", pues fue el modelo utilizado para crear los archivos de la conexión para ASIBOT y observar si con estos archivos se podía realizar de forma correcta la unión.

  Tras la descarga de los archivos [66] y lanzamiento de la orden correspondiente, se observó que la advertencia asociada al *controller_manager* seguía apareciendo (ver Figura 4.20), además de algunas asociadas al archivo .xacro utilizado para definir el modelo, pues debía de estar atrasado con respecto a la versión de ROS utilizada en este TFG. Ante esta situación se consideró una vía sin salida.

- **Estudio de un modelo más simple**
  Puesto que el programa se quedaba bloqueado de forma parecida a como sucedía con la librería OMPL, problema explicado en el punto 4.2.2, se decidió probar con un modelo más sencillo de ASIBOT para comprobar si era problema de nuevo, del tamaño de los archivo.

  Ya que en este caso no se disponía de archivos en formato STL de menor tamaño, se decidió crear un modelo que contuviese solo el "asibot__link0" y el "asibot__link1". Tras la creación de los archivos asociados al nuevo modelo, se observó una vez más que surgía el problema asociado al *controller_manager*, por lo que esta vía tampoco aportó ninguna solución.

- **Nuevas configuraciones con MoveIt**
  Con el objetivo de buscar algún tipo de cambio en la salida se definió ASIBOT por medio de otras configuraciones en MoveIt, sin embargo, en el mejor de los casos se obtuvo de nuevo la salida sin ningún tipo de advertencia o error, como se mostraba en la Figura 4.24.

Aunque se fueron subsanando los diferentes errores que fueron apareciendo hasta conseguir una ausencia total de los mismos y de las advertencias y, se abrieron diferentes vías de investigación, no fue posible solucionar el problema de bloqueo del programa y por lo tanto no se pudo realizar la conexión de forma satisfactoria. Por ese motivo se establece como posible mejora de este proyecto la realización de una investigación que permita realizar la unión de forma satisfactoria.

Capítulo 5

Resultados y conclusiones

5.1. Resultados del trabajo

A continuación se presentan los resultados finales del trabajo, es decir, las trayectorias que se han creado para el robot de asistencia ASIBOT.
Como ya se mencionó en el capítulo 1, ASIBOT es un robot orientado a la ayuda y asistencia de personas discapacitadas dentro del ambiente del hogar, pero también puede servir para ayudar a personas en rehabilitación. Por este motivo se han creado las trayectorias en estos dos campos. En total se han creado tres trayectorias, dos en entornos domésticos (cocina y baño) y una tercera de rehabilitación. Cada una se explicará en su sección correspondiente.

Antes de empezar con la explicación de cada una, es necesario puntualizar dos de aspectos: el primero es que puesto que cada una de las trayectorias utiliza una herramienta diferente en el extremo exterior, se crearon tres archivos .urdf distintos para no tener que estar modificándolos de forma continua, como se explicó en la subsección 4.2.2. El segundo es que todas las trayectorias están divididas en dos subtrayectorias, pues no era posible simularlas de forma continua. Esto es debido a que se realizan varios movimientos en ellas, por lo que es necesario seleccionar puntos de inicio y fin diferentes.

5.1.1. Trayectoria rehabilitación

En esta trayectoria se quería crear un ejercicio de rehabilitación para los hombros. Este podría se utilizado tanto en personas en silla de ruedas, como en personas mayores o que hayan sufrido alguna lesión de este tipo.

Lo que se pretende es que el usuario tenga que alcanzar la garra de ASIBOT a diferentes alturas. En este caso se ha ido de más abajo a más arriba, pero podría desarrollarse al revés. En concreto se han definido tres alturas, por lo que es necesario realizar dos subtrayectorias para poder definir la trayectoria final.

Subtrayectoria 1

En esta se sitúa el robot en una posición inicial baja, en rosa en la Figura 5.1, y se lleva hasta una altura intermedia que representa el punto final, en azul también en la Figura 5.1.

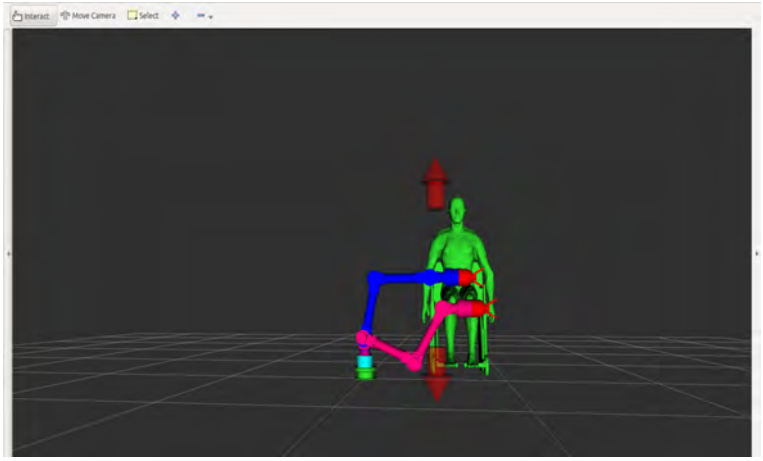

Figura 5.1: Posición inicial y final subtrayectoria 1 de rehabilitación

Puesto que en este documento no es posible mostrar correctamente la trayectoria (se presentarán vídeos de la misma en la defensa) se adjunta la Figura 5.2 en la que se puede ver el recorrido que realiza el robot.

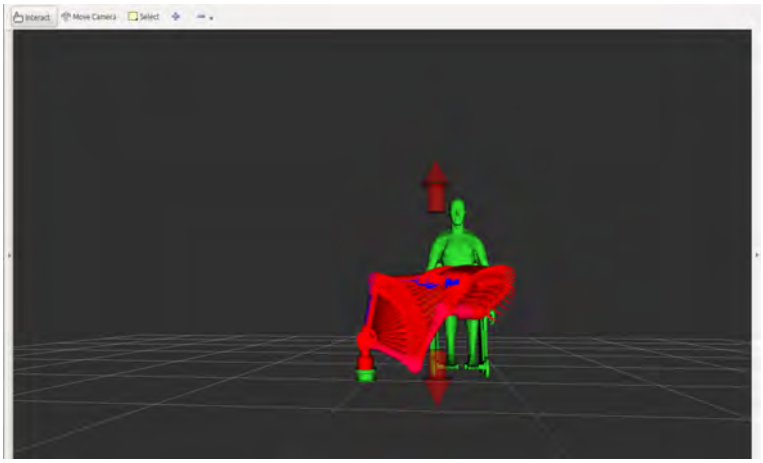

Figura 5.2: Recorrido realizado por ASIBOT en la subtrayectoria 1 de rehabilitación

Subtrayectoria 2

La segunda parte consiste en llegar a alcanzar una posición más elevada, partiendo de la posición intermedia. Por lo tanto la posición inicial que se muestra en rosa en la Figura 5.3, coincide con la posición final del punto anterior, el estado final aparece en la Figura 5.3 en azul y la Figura 5.4 muestra las distintas posiciones que ocupa el robot a lo largo del recorrido.

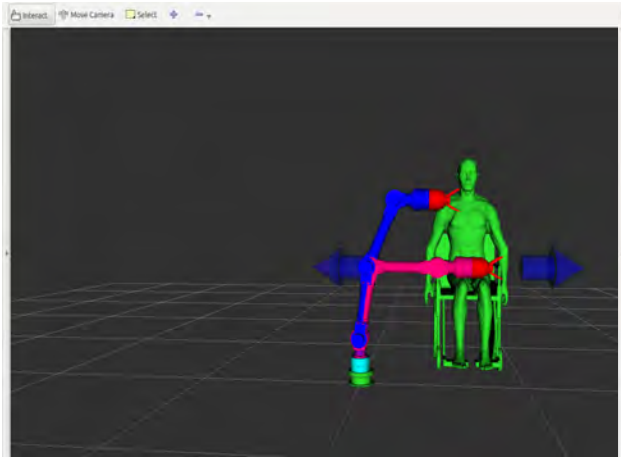

Figura 5.3: Posición inicial y final subtrayectoria 2 de rehabilitación

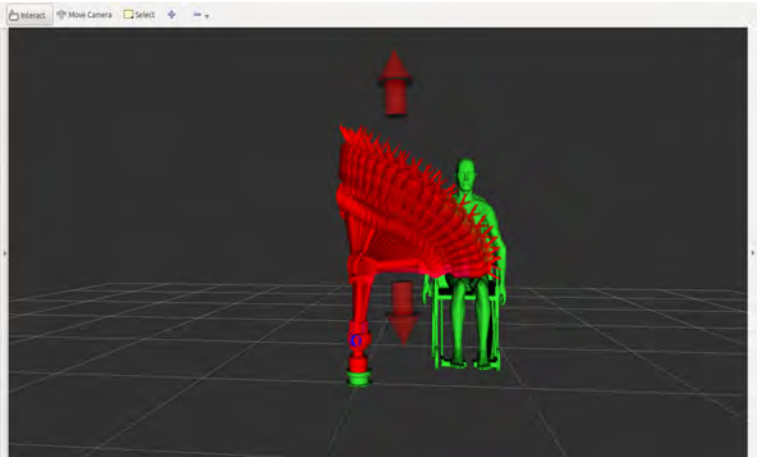

Figura 5.4: Recorrido realizado por ASIBOT en la subtrayectoria 2 de rehabilitación

5.1.2. Trayectoria en la cocina

El objetivo principal del proyecto ASIBOT es poder dotar de una mayor autonomía a personas con discapacidades (especialmente en sillas de ruedas) dentro de sus hogares, lo cual incluye ayudarles a realizar actividades tan cotidianas como coger un vaso de agua o comer. Esta trayectoria está orientada a estas tareas.

Esta en concreto representa la acción de coger agua del grifo y beberla. Para ello, puesto que la garra resulta poco precisa para este proceso (como se menciona en el capítulo 1), se utiliza la herramienta (*toolholder*) del vaso.

Este recorrido se divide en dos: el primero consiste en ir desde una posición inicial cualquiera hasta el fregadero y el segundo, desde el fregadero acercar el vaso a la persona para que beba.

Subtrayectoria 1

El recorrido de esta subtrayectoria consiste en llevar el vaso hasta el fregadero para coger agua.
En la Figura 5.5 se puede apreciar la posición inicial (en rosa) y final (en azul) de la trayectoria. Es decir, el robot se mueve desde el punto inicial que se le señala, en este caso se eligió un punto aleatorio, hasta la posición final, elegida por el usuario, debajo del grifo.

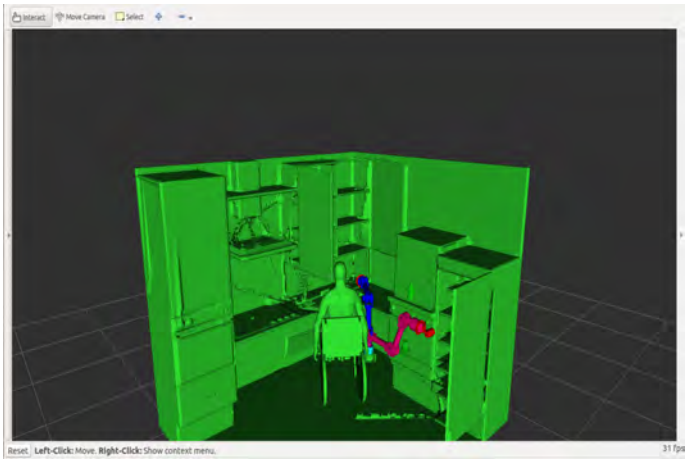

Figura 5.5: Posición inicial y final subtrayectoria 1 de la cocina

En la Figura 5.6 se puede observar el recorrido, mostrándose cada punto donde se encuentra el autómata a lo largo del movimiento.

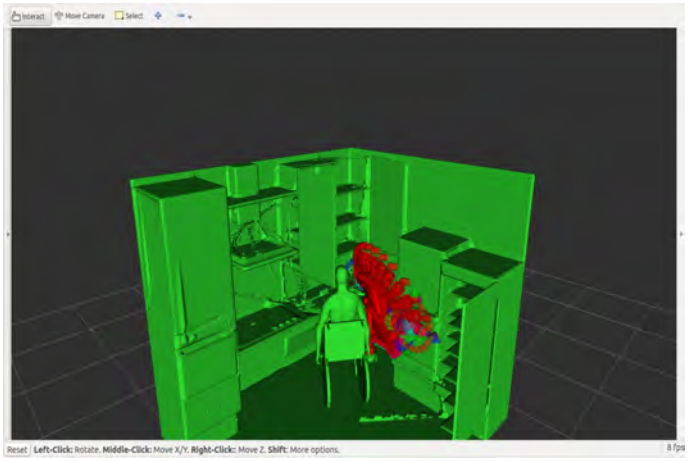

Figura 5.6: Recorrido realizado por ASIBOT en la subtrayectoria 1 de cocina

Subtrayectoria 2

La segunda parte de la trayectoria consiste en llevar el vaso hasta la persona para que beba. Como medida de seguridad se considera que es la persona la encargada de realizar el último movimiento, es decir, es el usuario el que tiene que acercarse al vaso para beber y no al revés, por eso se puede observar que este se deja a una cierta distancia.

De la misma forma que en la subtrayectoria anterior, la Figura 5.7 muestra la posición inicial, que es la final de la subtrayectoria 1 y la final de la subtrayectoria 2. En la Figura 5.8 es posible apreciar el movimiento en este segundo recorrido.

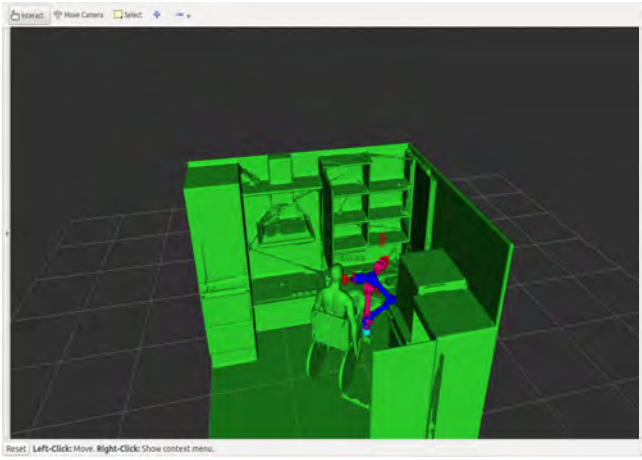

Figura 5.7: Posición inicial y final subtrayectoria 2 de cocina

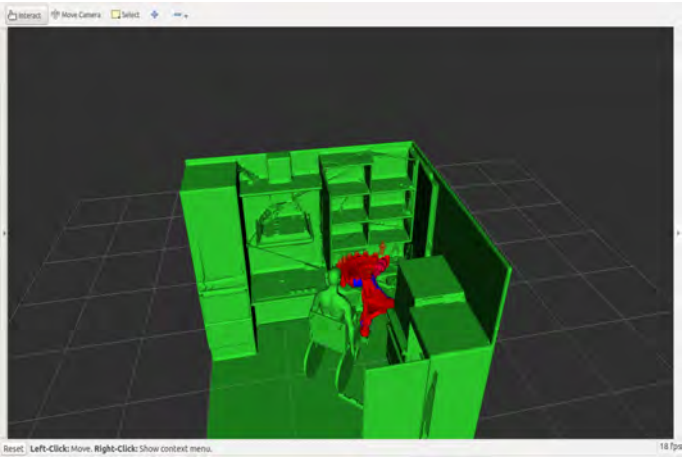

Figura 5.8: Recorrido realizado por ASIBOT en la subtrayectoria 2 de cocina

5.1.3. Trayectoria en el baño

El baño es otro entorno doméstico donde conseguir autonomía puede ser muy beneficioso para la persona, es por ello que se quiso diseñar esta trayectoria. En ella se representa el proceso de lavarse los dientes, por lo que fue necesario colocar una herramienta especial (*toolholder*) para sujetar el cepillo de dientes, que también se incluyó en la misma. Al igual que pasa con la de la cocina y la rehabilitación queda dividida en dos subtrayectorias.

Subtrayectoria 1

Desde una posición inicial aleatoria (se dejó que el programa Rviz la definiese), se viaja hasta la posición final debajo del grifo del lavabo. Estas dos posiciones pueden observase en la Figura 5.9, en rosa y azul respectivamente.

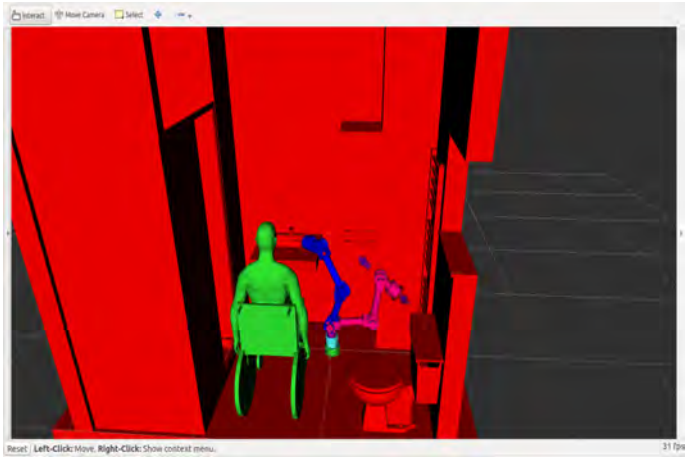

Figura 5.9: Posición inicial y final subtrayectoria 1 del baño

De nuevo para que resulte más visual el recorrido que hace la trayectoria, se muestra la Figura 5.10 con las distintas posiciones que ocupa el robot a lo largo de la misma.

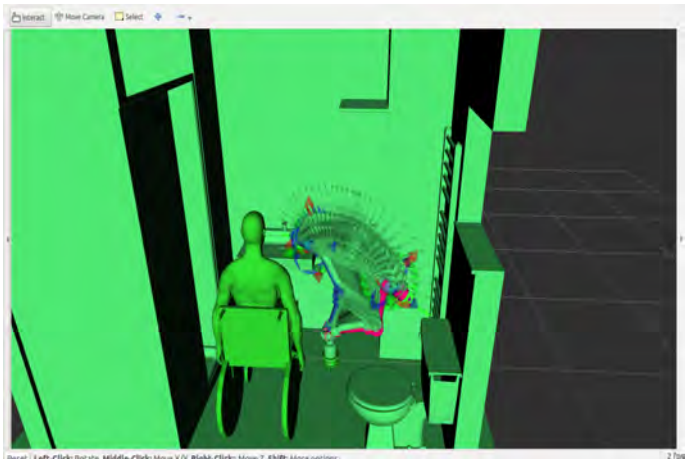

Figura 5.10: Recorrido realizado por ASIBOT en la subtrayectoria 1 del baño

Subtrayectoria 2

Una vez que se ha echado agua en el cepillo se puede realizar la acción de cepillarse los dientes. Para ello, en esta segunda subtrayectoria se hace un recorrido desde la posición inicial, que coincide con la final de la anterior subtrayectoria, hasta un punto cercano a la boca del usuario para que este ya pueda realizar el movimiento del cepillado. La Figura 5.11 representa las posiciones inicial y final, en rosa y azul respectivamente, que en este caso no distan mucho la una de la otra.

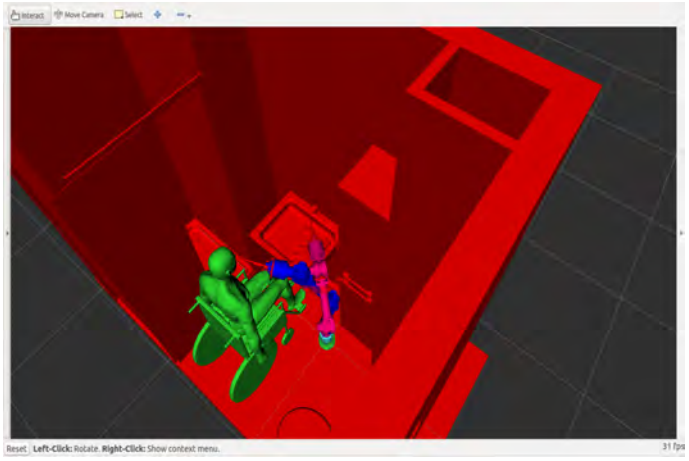

Figura 5.11: Posición inicial y final subtrayectoria 2 del baño

El recorrido de este segundo tramo es más corto y por lo tanto menos apreciable, pero se presenta igualmente en la Figura 5.12.

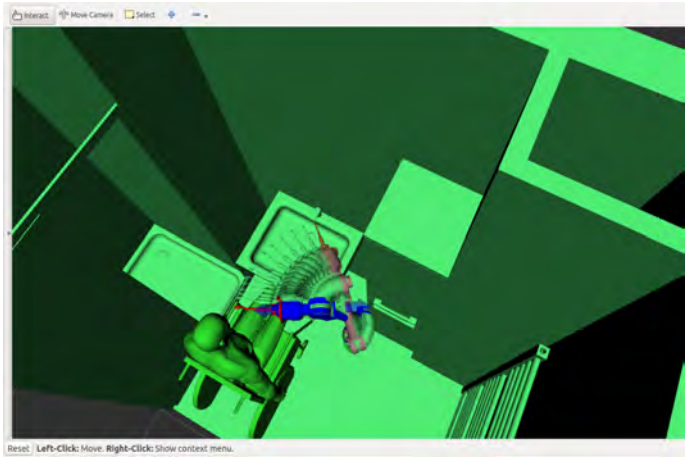

Figura 5.12: Recorrido realizado por ASIBOT en la subtrayectoria 2 del baño

5.2. Análisis crítico de los resultados

Una vez que se han creado las trayectorías, se pueden analizar los puntos fuertes y débiles de esta simulación.

1. **Puntos fuertes**
   Como se puede observar en las figuras de la sección 5.1 esta simulación permite crear y estudiar cualquier trayectoria deseada, de una forma sencilla, eficiente y económica. Permitiendo ahorrar mucho tiempo y dinero, pues hace prescindible la creación de entornos de prueba.

   Otra de las ventajas es el aumento de la seguridad, tanto de los operarios como de la estructura del propio robot, pues si algo mal en la simulación el entorno real no se ve afectado.

   También se puede destacar, que se ha conseguido una migración exitosa de los archivos definidos en formato .xml (utilizado en OpenRave) a .sdf y .urdf. Consiguiéndose de esta forma actualizar el entorno de trabajo y los modelos de ASIBOT, al simulador (Gazebo) más utilizado actualmente en robótica.

2. **Puntos débiles**
   Aunque es posible guardar los entornos como se mencionó en la subsección 4.2.2, cada vez que se abre el programa Rviz es necesario cargar de nuevo el entorno. Además, puesto que no fue posible realizar la conexión con Gazebo no se pueden, de momento, guardar las trayectorias, teniéndose que crear de nuevo cada vez que se entra en el programa.

5.3. Conclusiones

El objetivo de este Trabajo de Fin de Grado era la migración de los archivos de definición del robot asistencial ASIBOT al entorno de Gazebo y la creación de trayectorias por medio del programa MoveIt, que precisaba de dichos archivos, para comprobar su viabilidad a la hora de implementarlas en el modelo real.

Se consiguió realizar una migración exitosa del modelo de OpenRave al nuevo desarrollado en Gazebo, y aunque no fue posible realizar la conexión entre Rviz y Gazebo, las trayectorias implementadas en el primero resultaron correctas y posibles, por lo que sería factible la implementación de las mismas en el modelo real de ASIBOT.

Capítulo 6

Mejoras y trabajos futuros

Con vistas a la utilización de Gazebo como nuevo marco para las simulaciones robóticas a realizar con ASIBOT y basado en el trabajo realizado en este TFG, se presentan una serie de mejoras.

- **Finalización de la conexión Rviz-Gazebo:** La mejora principal sería conseguir realizar la conexión entre ambos programas, de forma que fuese posible exportar las trayectorias al simulador y de esta forma guardarlas para poder realizar las pruebas un entorno real.

  Una posible forma de investigar esta conexión es crear nuevos archivos para ASIBOT que pesen menos que los que lo conforman actualmente, pues sigue siendo una de las hipótesis que se baraja como causante del bloqueo.

- **Definición de nuevos archivos de entorno:** Como se explicó en el capítulo 3 y se puede apreciar en las imágenes de la simulación, la malla del entorno de la cocina no quedó bien definida y presenta una imagen bastante defectuosa frente al resto de elementos. Del mismo modo el entorno del baño resultaba tan grande que daba problemas a la hora de introducirlo como elemento del entorno dentro de Rviz. Debido a estas características sería recomendable realizar una nueva definición de la malla de ambos archivos en formato STL, utilizando un programa de modelado 3D.

- **Realización de pruebas:** Puesto que es posible desarrollar tantas trayectorias como se deseen, el paso siguiente sería introducirlas en ASIBOT y realizar pruebas dentro del entorno real, bien en el Robotics Lab o incluso realizar pruebas con pacientes.

Apéndices

Apéndice A

Marco Regulador

Al tratarse de un proyecto puramente de simulación sobre un robot, las legislaciones legales o económicas podrán aplicarse al autómata en s, pero no a la simulación realizada sobre el mismo.
Además, tanto los softwares como los modelos en 3D y códigos de terceros utilizados son de uso público, teniendo ROS una licencia estándar de tres cláusulas BSD [69], que permite el uso de estos códigos externos. Los modelos y códigos se encuentran todos en repositorios de uso público [70].

Apéndice B

Entorno Socio-Económico

Puesto que se trata de un trabajo meramente de simulación, el impacto económico que puede tener viene dado por la cantidad de tiempo, esfuerzo y dinero que permite ahorrar, pues no es necesaria la creación de entornos físicos y reales para la prueba de trayectorias, además de que permite la creación de un mayor número de ellas en un menor espacio de tiempo y evita posibles daños tanto al robot como al usuario al ser primero testado en un entorno digital.

En caso de que el robot ASIBOT llegase a introducirse en el mercado y este sistema de simulación siguiese en vigencia, tendría un alto impacto social pues permitiría el diseño y prueba de cualquier trayectoria necesitada por el usuario.

B.1. Presupuesto

Al tratarse de un proyecto puramente de programación, en el cual los softwares utilizados son completamente gratuitos, no existen costes materiales y todo el presupuesto viene determinado por la mano de obra. La Tabla B.1 muestra el desglose de los costes.

Descripción	**Unidades**	**Precio unitario**	**Coste**
Horas trabajadas	400 horas	10 €/hora	4000 €
Tutorías	25 horas		
Trabajo individual	375 horas		
Total neto			4000 €
IVA (21 %)			840 €
Total			**4840 €**

Tabla B.1: Presupuesto del proyecto

Apéndice C

Códigos

C.1. Código ASIBOT definido en OpenRave

```
<KinBody name="uc3m-asibot">
  <Body type="static" name="link0">
    <Geom type="trimesh">
      <rotationaxis>1 0 0 180</rotationaxis>
      <translation>0 0 -0.07</translation>
      <data>./ivs/machosn.iv</data>
      <render>./ivs/machosn.iv</render>
    </Geom>
    <Geom type="trimesh">
      <data>./ivs/ensam01sn.iv</data>
      <render>./ivs/ensam01sn.iv</render>
    </Geom>
    <mass type="sphere">
      <total>1</total>
      <radius>1</radius>
    </mass>
  </Body>
  <Body name="link1">
    <offsetfrom>link0</offsetfrom>
    <Geom type="trimesh">
      <translation>0 0 0.12</translation>
      <data>./ivs/ensam02sn.iv</data>
      <render>./ivs/ensam02sn.iv</render>
    </Geom>
    <mass type="sphere">
      <total>1</total>
      <radius>1</radius>
    </mass>
  </Body>
  <Joint name="M1" type="hinge">
    <offsetfrom>link0</offsetfrom>
    <body>link0</body>
    <body>link1</body>
    <axis>0 0 1</axis>
    <lostop>-180</lostop>
    <histop>180</histop>
    <maxvel>2</maxvel>
  </Joint>
  <Body name="link2">
    <offsetfrom>link0</offsetfrom>
    <Geom type="trimesh">
      <translation>0 0.0145 0.365</translation>
      <data>./ivs/ensam03sn.iv</data>
      <render>./ivs/ensam03sn.iv</render>
    </Geom>
    <mass type="sphere">
      <total>1</total>
      <radius>1</radius>
    </mass>
  </Body>
  <Joint name="M2" type="hinge">
    <offsetfrom>link1</offsetfrom>
    <body>link1</body>
    <body>link2</body>
    <axis>0 1 0</axis>
    <anchor>0 0 0.165</anchor>
```

```
    <lostop>-135</lostop>
    <histop>135</histop>
    <maxvel>2</maxvel>
  </Joint>
  <Body name="link3">
    <offsetfrom>link0</offsetfrom>
    <Geom type="trimesh">
      <translation>0 0.0145 0.76</translation>
      <data>./ivs/ensam04sn.iv</data>
      <render>./ivs/ensam04sn.iv</render>
    </Geom>
    <mass type="sphere">
      <total>1</total>
      <radius>1</radius>
    </mass>
  </Body>
  <Joint name="M3" type="hinge">
    <offsetfrom>link2</offsetfrom>
    <body>link2</body>
    <body>link3</body>
    <axis>0 1 0</axis>
    <anchor>0 0 0.563</anchor>
    <lostop>-135</lostop>
    <histop>135</histop>
    <maxvel>2</maxvel>
  </Joint>
  <Body name="link4">
    <offsetfrom>link0</offsetfrom>
    <Geom type="trimesh">
      <rotationaxis>1 0 0 180</rotationaxis>
      <translation>0 0.001 1.007</translation>
      <data>./ivs/ensam02sn.iv</data>
      <render>./ivs/ensam02sn.iv</render>
    </Geom>
    <mass type="sphere">
      <total>1</total>
      <radius>1</radius>
    </mass>
  </Body>
  <Joint name="M4" type="hinge">
    <offsetfrom>link3</offsetfrom>
    <body>link3</body>
    <body>link4</body>
    <axis>0 1 0</axis>
    <anchor>0 0 0.962</anchor>
    <lostop>-135</lostop>
    <histop>135</histop>
    <maxvel>2</maxvel>
  </Joint>
  <Body name="link5">
    <offsetfrom>link0</offsetfrom>
    <Geom type="trimesh">
      <rotationaxis>1 0 0 180</rotationaxis>
      <translation>0 0 1.127</translation>
      <data>./ivs/ensam01sn.iv</data>
      <render>./ivs/ensam01sn.iv</render>
    </Geom>
    <Geom type="trimesh">
      <translation>0 0 1.197</translation>
      <!--data>./ivs/machosn.iv</data-->
      <render>./ivs/machosn.iv</render>
    </Geom>
    <Geom type="trimesh">
      <rotationaxis>0 0 1 90</rotationaxis>
      <translation>0.006 0 1.295</translation>
      <Data>./ivs/pinzapas_s.iv 1</Data>
      <Render>./ivs/pinzapas_s.iv 1</Render>
    </Geom>
    <mass type="sphere">
      <total>0.33</total>
      <radius>1</radius>
    </mass>
  </Body>
  <Joint name="M5" type="hinge">
    <offsetfrom>link3</offsetfrom>
    <body>link4</body>
    <body>link5</body>
    <axis>0 0 1</axis>
    <anchor>0 0 1</anchor>
    <lostop>-180</lostop>
    <histop>180</histop>
    <maxvel>2</maxvel>
  </Joint>
  <Body name="link6">
    <offsetfrom>link0</offsetfrom>
    <Geom type="trimesh">
      <rotationaxis>0 0 1 90</rotationaxis>
      <translation>0.0055 0.001 1.295</translation>
      <Data>./ivs/pinzapas_pl.iv 1</Data>
      <Render>./ivs/pinzapas_pl.iv 1</Render>
```

```
    </Geom>
    <mass type="sphere">
      <total>0.1</total>
      <radius>1</radius>
    </mass>
  </Body>
  <Joint name="P1" type="hinge">
    <offsetfrom>link5</offsetfrom>
    <body>link5</body>
    <body>link6</body>
    <axis>1 0 0</axis>
    <anchor>0 0 1.342</anchor>
    <lostop>-180</lostop>
    <histop>180</histop>
    <maxvel>2</maxvel>
  </Joint>
  <Body name="link7">
    <offsetfrom>link0</offsetfrom>
    <Geom type="trimesh">
      <rotationaxis>0 0 1 90</rotationaxis>
      <translation>0.0061 -0.001 1.295</translation>
      <Data>./ivs/pinzapas_pr.iv 1</Data>
      <Render>./ivs/pinzapas_pr.iv 1</Render>
    </Geom>
    <mass type="sphere">
      <total>0.1</total>
      <radius>1</radius>
    </mass>
  </Body>
  <Joint name="P2" type="hinge">
    <offsetfrom>link5</offsetfrom>
    <body>link5</body>
    <body>link7</body>
    <axis>1 0 0</axis>
    <anchor>0 0 1.342</anchor>
    <lostop>-180</lostop>
    <histop>180</histop>
    <maxvel>2</maxvel>
  </Joint>

<!-- no joint limits -->

</KinBody>
```

C.2. Código ASIBOT definido en Gazebo

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <model name='asibot'>
    <link name='link0'>
      <pose frame=''>0 0 0 0 -0 0</pose>
      <inertial>
        <mass>1</mass>
        <inertia>
          <ixx>1</ixx>
          <ixy>0</ixy>
          <iyy>1</iyy>
          <ixz>0</ixz>
          <iyz>0</iyz>
          <izz>1</izz>
        </inertia>
        <pose frame=''>0 0 0 0 -0 0</pose>
      </inertial>
      <self_collide>0</self_collide>
      <kinematic>0</kinematic>
      <visual name='visual'>
        <pose frame=''>0 0 0 0 -0 0</pose>
        <geometry>
          <mesh>
            <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam01s.stl</uri>
            <scale>0.001 0.001 0.001</scale>
          </mesh>
        </geometry>
        <transparency>0</transparency>
        <cast_shadows>1</cast_shadows>
        <material>
          <shader type='vertex'>
            <normal_map>__default__</normal_map>
          </shader>
        </material>
      </visual>
      <visual name='visual2'>
        <pose frame=''>0 0 0.015 3.14159 -0 0</pose>
```

```
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/macho.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <cast_shadows>1</cast_shadows>
      <transparency>0</transparency>
      <material>
        <shader type='vertex'>
          <normal_map>__default__</normal_map>
        </shader>
      </material>
    </visual>
    <collision name='collision'>
      <laser_retro>0</laser_retro>
      <max_contacts>10</max_contacts>
      <pose frame=''>0 0 0 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam01s.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
          <torsional>
            <coefficient>1</coefficient>
            <patch_radius>0</patch_radius>
            <surface_radius>0</surface_radius>
            <use_patch_radius>1</use_patch_radius>
            <ode>
              <slip>0</slip>
            </ode>
          </torsional>
        </friction>
        <bounce>
          <restitution_coefficient>0</restitution_coefficient>
          <threshold>1e+06</threshold>
        </bounce>
        <contact>
          <collide_without_contact>0</collide_without_contact>
          <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
          <collide_bitmask>1</collide_bitmask>
          <ode>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
            <max_vel>0.01</max_vel>
            <min_depth>0</min_depth>
          </ode>
          <bullet>
            <split_impulse>1</split_impulse>
            <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
          </bullet>
        </contact>
      </surface>
    </collision>
    <collision name='collision2'>
      <laser_retro>0</laser_retro>
      <max_contacts>10</max_contacts>
      <pose frame=''>0 0 0.015 3.14159 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/macho.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
          <torsional>
```

```
            <coefficient>1</coefficient>
            <patch_radius>0</patch_radius>
            <surface_radius>0</surface_radius>
            <use_patch_radius>1</use_patch_radius>
            <ode>
              <slip>0</slip>
            </ode>
          </torsional>
        </friction>
        <bounce>
          <restitution_coefficient>0</restitution_coefficient>
          <threshold>1e+06</threshold>
        </bounce>
        <contact>
          <collide_without_contact>0</collide_without_contact>
          <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
          <collide_bitmask>1</collide_bitmask>
          <ode>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
            <max_vel>0.01</max_vel>
            <min_depth>0</min_depth>
          </ode>
          <bullet>
            <split_impulse>1</split_impulse>
            <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
          </bullet>
        </contact>
      </surface>
    </collision>
  </link>
  <link name='link1'>
    <pose frame=''>0 0 0 0 -0 0</pose>
    <inertial>
      <mass>1</mass>
      <inertia>
        <ixx>1</ixx>
        <ixy>0</ixy>
        <iyy>1</iyy>
        <ixz>0</ixz>
        <iyz>0</iyz>
        <izz>1</izz>
      </inertia>
      <pose frame=''>0 0 0 0 -0 0</pose>
    </inertial>
    <self_collide>0</self_collide>
    <kinematic>0</kinematic>
    <visual name='visual'>
      <pose frame=''>0 0 0.012 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam02.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <transparency>0</transparency>
      <cast_shadows>1</cast_shadows>
      <material>
        <shader type='vertex'>
          <normal_map>__default__</normal_map>
        </shader>
      </material>
    </visual>
    <collision name='collision'>
      <laser_retro>0</laser_retro>
      <max_contacts>10</max_contacts>
      <pose frame=''>0 0 0.012 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam02.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
          <torsional>
            <coefficient>1</coefficient>
```

```
            <patch_radius>0</patch_radius>
            <surface_radius>0</surface_radius>
            <use_patch_radius>1</use_patch_radius>
            <ode>
              <slip>0</slip>
            </ode>
          </torsional>
        </friction>
        <bounce>
          <restitution_coefficient>0</restitution_coefficient>
          <threshold>1e+06</threshold>
        </bounce>
        <contact>
          <collide_without_contact>0</collide_without_contact>
          <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
          <collide_bitmask>1</collide_bitmask>
          <ode>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
            <max_vel>0.01</max_vel>
            <min_depth>0</min_depth>
          </ode>
          <bullet>
            <split_impulse>1</split_impulse>
            <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
          </bullet>
        </contact>
      </surface>
    </collision>
  </link>
  <link name='link2'>
    <pose frame=''>0 0 0.165 0 -0 0</pose>
    <inertial>
      <mass>1</mass>
      <inertia>
        <ixx>1</ixx>
        <ixy>0</ixy>
        <iyy>1</iyy>
        <ixz>0</ixz>
        <iyz>0</iyz>
        <izz>1</izz>
      </inertia>
      <pose frame=''>0 0 0.165 0 -0 0</pose>
    </inertial>
    <self_collide>0</self_collide>
    <kinematic>0</kinematic>
    <visual name='visual'>
      <pose frame=''>0 0.0145 -0.06 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam03.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <transparency>0</transparency>
      <cast_shadows>1</cast_shadows>
      <material>
        <shader type='vertex'>
          <normal_map>__default__</normal_map>
        </shader>
      </material>
    </visual>
    <collision name='collision'>
      <laser_retro>0</laser_retro>
      <max_contacts>10</max_contacts>
      <pose frame=''>0 0.0145 -0.06 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam03.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
          <torsional>
            <coefficient>1</coefficient>
            <patch_radius>0</patch_radius>
```

```
            <surface_radius>0</surface_radius>
            <use_patch_radius>1</use_patch_radius>
            <ode>
              <slip>0</slip>
            </ode>
          </torsional>
        </friction>
        <bounce>
          <restitution_coefficient>0</restitution_coefficient>
          <threshold>1e+06</threshold>
        </bounce>
        <contact>
          <collide_without_contact>0</collide_without_contact>
          <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
          <collide_bitmask>1</collide_bitmask>
          <ode>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
            <max_vel>0.01</max_vel>
            <min_depth>0</min_depth>
          </ode>
          <bullet>
            <split_impulse>1</split_impulse>
            <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
          </bullet>
        </contact>
      </surface>
    </collision>
  </link>
  <link name='link3'>
    <pose frame=''>0 0 0.56299 0 -0 0</pose>
    <inertial>
      <mass>1</mass>
      <inertia>
        <ixx>1</ixx>
        <ixy>0</ixy>
        <iyy>1</iyy>
        <ixz>0</ixz>
        <iyz>0</iyz>
        <izz>1</izz>
      </inertia>
      <pose frame=''>0 0 0.56299 0 -0 0</pose>
    </inertial>
    <self_collide>0</self_collide>
    <kinematic>0</kinematic>
    <visual name='visual'>
      <pose frame=''>0 0.0145 -0.06 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam04.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <transparency>0</transparency>
      <cast_shadows>1</cast_shadows>
      <material>
        <shader type='vertex'>
          <normal_map>__default__</normal_map>
        </shader>
      </material>
    </visual>
    <collision name='collision'>
      <laser_retro>0</laser_retro>
      <max_contacts>10</max_contacts>
      <pose frame=''>0 0.0145 -0.06 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam04.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
          <torsional>
            <coefficient>1</coefficient>
            <patch_radius>0</patch_radius>
            <surface_radius>0</surface_radius>
```

```
            <use_patch_radius>1</use_patch_radius>
            <ode>
              <slip>0</slip>
            </ode>
          </torsional>
        </friction>
        <bounce>
          <restitution_coefficient>0</restitution_coefficient>
          <threshold>1e+06</threshold>
        </bounce>
        <contact>
          <collide_without_contact>0</collide_without_contact>
          <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
          <collide_bitmask>1</collide_bitmask>
          <ode>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
            <max_vel>0.01</max_vel>
            <min_depth>0</min_depth>
          </ode>
          <bullet>
            <split_impulse>1</split_impulse>
            <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
          </bullet>
        </contact>
      </surface>
    </collision>
  </link>
  <link name='link4'>
    <pose frame=''>0 0 0.962 0 -0 0</pose>
    <inertial>
      <mass>1</mass>
      <inertia>
        <ixx>1</ixx>
        <ixy>0</ixy>
        <iyy>1</iyy>
        <ixz>0</ixz>
        <iyz>0</iyz>
        <izz>1</izz>
      </inertia>
      <pose frame=''>0 0 0.962 0 -0 0</pose>
    </inertial>
    <self_collide>0</self_collide>
    <kinematic>0</kinematic>
    <visual name='visual'>
      <pose frame=''>-0.0012 0.001 0.14 3.14159 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam02.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <transparency>0</transparency>
      <cast_shadows>1</cast_shadows>
      <material>
        <shader type='vertex'>
          <normal_map>__default__</normal_map>
        </shader>
      </material>
    </visual>
    <collision name='collision'>
      <laser_retro>0</laser_retro>
      <max_contacts>10</max_contacts>
      <pose frame=''>-0.0012 0.001 0.14 3.14159 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam02.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
          <torsional>
            <coefficient>1</coefficient>
            <patch_radius>0</patch_radius>
            <surface_radius>0</surface_radius>
            <use_patch_radius>1</use_patch_radius>
```

```
            <ode>
              <slip>0</slip>
            </ode>
          </torsional>
        </friction>
        <bounce>
          <restitution_coefficient>0</restitution_coefficient>
          <threshold>1e+06</threshold>
        </bounce>
        <contact>
          <collide_without_contact>0</collide_without_contact>
          <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
          <collide_bitmask>1</collide_bitmask>
          <ode>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
            <max_vel>0.01</max_vel>
            <min_depth>0</min_depth>
          </ode>
          <bullet>
            <split_impulse>1</split_impulse>
            <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
          </bullet>
        </contact>
      </surface>
    </collision>
  </link>
  <link name='link5'>
    <pose frame=''>0 0 1 0 -0 0</pose>
    <inertial>
      <mass>0.33</mass>
      <inertia>
        <ixx>1</ixx>
        <ixy>0</ixy>
        <iyy>1</iyy>
        <ixz>0</ixz>
        <iyz>0</iyz>
        <izz>1</izz>
      </inertia>
      <pose frame=''>0 0 1 0 -0 0</pose>
    </inertial>
    <self_collide>0</self_collide>
    <kinematic>0</kinematic>
    <gravity>1</gravity>
    <visual name='ModelPreview_4::link5::visual_5'>
      <pose frame=''>-0.0012 0 0.097 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/macho.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <material>
        <lighting>1</lighting>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <shader type='vertex'>
          <normal_map>__default__</normal_map>
        </shader>
        <ambient>0.3 0.3 0.3 1</ambient>
        <diffuse>0.7 0.7 0.7 1</diffuse>
        <specular>0.01 0.01 0.01 1</specular>
        <emissive>0 0 0 1</emissive>
      </material>
      <transparency>0</transparency>
      <cast_shadows>1</cast_shadows>
    </visual>
    <visual name='ModelPreview_4::link5::visual_4'>
      <pose frame=''>-0.0012 0 0.157 0 -0 1.5708</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ToolholderVaso.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <material>
        <lighting>1</lighting>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <shader type='vertex'>
```

```
          <normal_map>__default__</normal_map>
        </shader>
        <ambient>0.3 0.3 0.3 1</ambient>
        <diffuse>0.7 0.7 0.7 1</diffuse>
        <specular>0.01 0.01 0.01 1</specular>
        <emissive>0 0 0 1</emissive>
      </material>
      <transparency>0</transparency>
      <cast_shadows>1</cast_shadows>
    </visual>
    <visual name='visual'>
      <pose frame=''>-0.0012 0 0.115 3.14159 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam01s.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <material>
        <lighting>1</lighting>
        <script>
          <uri>file://media/materials/scripts/gazebo.material</uri>
          <name>Gazebo/Grey</name>
        </script>
        <shader type='vertex'>
          <normal_map>__default__</normal_map>
        </shader>
        <ambient>0.3 0.3 0.3 1</ambient>
        <diffuse>0.7 0.7 0.7 1</diffuse>
        <specular>0.01 0.01 0.01 1</specular>
        <emissive>0 0 0 1</emissive>
      </material>
      <transparency>0</transparency>
      <cast_shadows>1</cast_shadows>
    </visual>
    <collision name='collision_2'>
      <laser_retro>0</laser_retro>
      <max_contacts>10</max_contacts>
      <pose frame=''>-0.025 0 0.067 3.14159 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam01s.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
          <torsional>
            <coefficient>1</coefficient>
            <patch_radius>0</patch_radius>
            <surface_radius>0</surface_radius>
            <use_patch_radius>1</use_patch_radius>
            <ode>
              <slip>0</slip>
            </ode>
          </torsional>
        </friction>
        <bounce>
          <restitution_coefficient>0</restitution_coefficient>
          <threshold>1e+06</threshold>
        </bounce>
        <contact>
          <collide_without_contact>0</collide_without_contact>
          <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
          <collide_bitmask>1</collide_bitmask>
          <ode>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
            <max_vel>0.01</max_vel>
            <min_depth>0</min_depth>
          </ode>
          <bullet>
            <split_impulse>1</split_impulse>
            <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
          </bullet>
        </contact>
      </surface>
```

```
    </collision>
    <collision name='collision_3'>
      <laser_retro>0</laser_retro>
      <max_contacts>10</max_contacts>
      <pose frame=''>-0.0012 0 0.157 0 -0 1.5708</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ToolholderVaso.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
          <torsional>
            <coefficient>1</coefficient>
            <patch_radius>0</patch_radius>
            <surface_radius>0</surface_radius>
            <use_patch_radius>1</use_patch_radius>
            <ode>
              <slip>0</slip>
            </ode>
          </torsional>
        </friction>
        <bounce>
          <restitution_coefficient>0</restitution_coefficient>
          <threshold>1e+06</threshold>
        </bounce>
        <contact>
          <collide_without_contact>0</collide_without_contact>
          <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
          <collide_bitmask>1</collide_bitmask>
          <ode>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
            <max_vel>0.01</max_vel>
            <min_depth>0</min_depth>
          </ode>
          <bullet>
            <split_impulse>1</split_impulse>
            <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
          </bullet>
        </contact>
      </surface>
    </collision>
    <collision name='collision'>
      <laser_retro>0</laser_retro>
      <max_contacts>10</max_contacts>
      <pose frame=''>-0.0012 0 0.097 0 -0 0</pose>
      <geometry>
        <mesh>
          <uri>/home/celia/Documentos/tfg/Archivos_stl/stl_parque/macho.stl</uri>
          <scale>0.001 0.001 0.001</scale>
        </mesh>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>1</mu>
            <mu2>1</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
          <torsional>
            <coefficient>1</coefficient>
            <patch_radius>0</patch_radius>
            <surface_radius>0</surface_radius>
            <use_patch_radius>1</use_patch_radius>
            <ode>
              <slip>0</slip>
            </ode>
          </torsional>
        </friction>
        <bounce>
          <restitution_coefficient>0</restitution_coefficient>
          <threshold>1e+06</threshold>
        </bounce>
```

```
        <contact>
          <collide_without_contact>0</collide_without_contact>
          <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
          <collide_bitmask>1</collide_bitmask>
          <ode>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
            <max_vel>0.01</max_vel>
            <min_depth>0</min_depth>
          </ode>
          <bullet>
            <split_impulse>1</split_impulse>
            <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
            <soft_cfm>0</soft_cfm>
            <soft_erp>0.2</soft_erp>
            <kp>1e+13</kp>
            <kd>1</kd>
          </bullet>
        </contact>
      </surface>
    </collision>
  </link>
  <joint name='M1' type='revolute'>
    <parent>link0</parent>
    <child>link1</child>
    <pose frame=''>0 0 0 0 -0 0</pose>
    <axis>
      <xyz>0 0 1</xyz>
      <use_parent_model_frame>0</use_parent_model_frame>
      <limit>
        <lower>-3.1416</lower>
        <upper>3.1416</upper>
        <effort>-1</effort>
        <velocity>2</velocity>
      </limit>
      <dynamics>
        <spring_reference>0</spring_reference>
        <spring_stiffness>0</spring_stiffness>
        <damping>0</damping>
        <friction>0</friction>
      </dynamics>
    </axis>
    <physics>
      <ode>
        <limit>
          <cfm>0</cfm>
          <erp>0.2</erp>
        </limit>
        <suspension>
          <cfm>0</cfm>
          <erp>0.2</erp>
        </suspension>
      </ode>
    </physics>
  </joint>
  <joint name='M2' type='revolute'>
    <parent>link1</parent>
    <child>link2</child>
    <pose frame=''>0 0 0.165 0 -0 0</pose>
    <axis>
      <xyz>0 1 0</xyz>
      <use_parent_model_frame>0</use_parent_model_frame>
      <limit>
        <lower>-2.3562</lower>
        <upper>2.3562</upper>
        <effort>-1</effort>
        <velocity>2</velocity>
      </limit>
      <dynamics>
        <spring_reference>0</spring_reference>
        <spring_stiffness>0</spring_stiffness>
        <damping>0</damping>
        <friction>0</friction>
      </dynamics>
    </axis>
    <physics>
      <ode>
        <limit>
          <cfm>0</cfm>
          <erp>0.2</erp>
        </limit>
        <suspension>
          <cfm>0</cfm>
          <erp>0.2</erp>
        </suspension>
      </ode>
    </physics>
  </joint>
```

```
<joint name='M3' type='revolute'>
  <parent>link2</parent>
  <child>link3</child>
  <pose frame=''>0 0 0.563 0 -0 0</pose>
  <axis>
    <xyz>0 1 0</xyz>
    <use_parent_model_frame>0</use_parent_model_frame>
    <limit>
      <lower>-2.3562</lower>
      <upper>2.3562</upper>
      <effort>-1</effort>
      <velocity>2</velocity>
    </limit>
    <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
      <damping>0</damping>
      <friction>0</friction>
    </dynamics>
  </axis>
  <physics>
    <ode>
      <limit>
        <cfm>0</cfm>
        <erp>0.2</erp>
      </limit>
      <suspension>
        <cfm>0</cfm>
        <erp>0.2</erp>
      </suspension>
    </ode>
  </physics>
</joint>
<joint name='M4' type='revolute'>
  <parent>link3</parent>
  <child>link4</child>
  <pose frame=''>0 0 0.962 0 -0 0</pose>
  <axis>
    <xyz>0 1 0</xyz>
    <use_parent_model_frame>0</use_parent_model_frame>
    <limit>
      <lower>-2.3562</lower>
      <upper>2.3562</upper>
      <effort>-1</effort>
      <velocity>2</velocity>
    </limit>
    <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
      <damping>0</damping>
      <friction>0</friction>
    </dynamics>
  </axis>
  <physics>
    <ode>
      <limit>
        <cfm>0</cfm>
        <erp>0.2</erp>
      </limit>
      <suspension>
        <cfm>0</cfm>
        <erp>0.2</erp>
      </suspension>
    </ode>
  </physics>
</joint>
<joint name='M5' type='revolute'>
  <parent>link4</parent>
  <child>link5</child>
  <pose frame=''>0 0 1 0 -0 0</pose>
  <axis>
    <xyz>0 0 1</xyz>
    <use_parent_model_frame>0</use_parent_model_frame>
    <limit>
      <lower>-3.1416</lower>
      <upper>3.1416</upper>
      <effort>-1</effort>
      <velocity>2</velocity>
    </limit>
    <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
      <damping>0</damping>
      <friction>0</friction>
    </dynamics>
  </axis>
  <physics>
    <ode>
      <limit>
        <cfm>0</cfm>
```

```
          <erp>0.2</erp>
        </limit>
        <suspension>
          <cfm>0</cfm>
          <erp>0.2</erp>
        </suspension>
      </ode>
    </physics>
  </joint>
  <static>0</static>
  <allow_auto_disable>1</allow_auto_disable>
 </model>
</sdf>
```

C.3. Código cocina definido en OpenRave

```
<KinBody name="cocina">
  <RotationAxis>1 0 0 0</RotationAxis>
  <Body type="static">
    <Geom type="trimesh">
      <Data>./ivs/cocina.iv</Data>
      <Render>./ivs/cocina.iv</Render>
    </Geom>
  </Body>
</KinBody>
```

C.4. Código cocina definido en Gazebo

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <model name='cocina'>
    <link name='link'>
      <pose frame=''>0 0 0 0 -0 0</pose>
      <inertial>
        <mass>1</mass>
        <pose frame=''>0 0 0 0 -0 0</pose>
        <inertia>
          <ixx>1</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>1</iyy>
          <iyz>0</iyz>
          <izz>1</izz>
        </inertia>
      </inertial>
      <self_collide>0</self_collide>
      <kinematic>0</kinematic>
      <gravity>1</gravity>
      <visual name='visual'>
        <geometry>
          <mesh>
            <uri>/home/celia/Documentos/tfg/Archivos_stl/girados/Modelos_girados/cocina.stl</uri>
          </mesh>
        </geometry>
        <pose frame=''>0 0 0 0 -0 0</pose>
        <transparency>0</transparency>
        <cast_shadows>1</cast_shadows>
        <material>
          <shader type='vertex'>
            <normal_map>__default__</normal_map>
          </shader>
        </material>
      </visual>
      <collision name='collision'>
        <laser_retro>0</laser_retro>
        <max_contacts>10</max_contacts>
        <pose frame=''>0 0 0 0 -0 0</pose>
        <geometry>
          <mesh>
            <uri>/home/celia/Documentos/tfg/Archivos_stl/girados/Modelos_girados/cocina.stl</uri>
            <scale>1 1 1</scale>
          </mesh>
        </geometry>
        <surface>
          <friction>
```

```
            <ode>
              <mu>1</mu>
              <mu2>1</mu2>
              <fdir1>0 0 0</fdir1>
              <slip1>0</slip1>
              <slip2>0</slip2>
            </ode>
            <torsional>
              <coefficient>1</coefficient>
              <patch_radius>0</patch_radius>
              <surface_radius>0</surface_radius>
              <use_patch_radius>1</use_patch_radius>
              <ode>
                <slip>0</slip>
              </ode>
            </torsional>
          </friction>
          <bounce>
            <restitution_coefficient>0</restitution_coefficient>
            <threshold>1e+06</threshold>
          </bounce>
          <contact>
            <collide_without_contact>0</collide_without_contact>
            <collide_without_contact_bitmask>1</collide_without_contact_bitmask>
            <collide_bitmask>1</collide_bitmask>
            <ode>
              <soft_cfm>0</soft_cfm>
              <soft_erp>0.2</soft_erp>
              <kp>1e+13</kp>
              <kd>1</kd>
              <max_vel>0.01</max_vel>
              <min_depth>0</min_depth>
            </ode>
            <bullet>
              <split_impulse>1</split_impulse>
              <split_impulse_penetration_threshold>-0.01</split_impulse_penetration_threshold>
              <soft_cfm>0</soft_cfm>
              <soft_erp>0.2</soft_erp>
              <kp>1e+13</kp>
              <kd>1</kd>
            </bullet>
          </contact>
        </surface>
      </collision>
    </link>
    <static>1</static>
    <allow_auto_disable>1</allow_auto_disable>
  </model>
</sdf>
```

C.5. Código ASIBOT garra definido en URDF

```
<?xml version="1.0" ?>
<robot name="asibot">

<!--joint M1-->
  <joint name="asibot__M1" type="revolute">
    <parent link="asibot__link0"/>
    <child link="asibot__link1"/>
    <origin rpy="0␣␣0␣␣0" xyz="0␣␣0␣␣0"/>
    <axis xyz="0␣␣0␣␣1"/>
    <limit effort="-1.0" lower="-3.1416" upper="3.1416" velocity="2.0"/>
  </joint>

<!--joint M2-->
  <joint name="asibot__M2" type="revolute">
    <parent link="asibot__link1"/>
    <child link="asibot__link2"/>
    <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣0␣␣␣␣0.165"/>
    <axis xyz="0␣␣1␣␣0"/>
    <limit effort="-1.0" lower="-2.3562" upper="2.3562" velocity="2.0"/>
  </joint>

<!--joint M3-->
  <joint name="asibot__M3" type="revolute">
    <parent link="asibot__link2"/>
    <child link="asibot__link3"/>
    <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣␣␣0␣␣␣␣␣␣␣0.3925"/>
    <axis xyz="0␣␣1␣␣0"/>
    <limit effort="-1.0" lower="-2.3562" upper="2.3562" velocity="2.0"/>
  </joint>

<!--joint M4-->
```

```
  <joint name="asibot__M4" type="revolute">
    <parent link="asibot__link3"/>
    <child link="asibot__link4"/>
    <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣␣␣0␣␣␣␣␣␣␣0.398"/>
    <axis xyz="0␣␣1␣␣0"/>
    <limit effort="-1.0" lower="-2.3562" upper="2.3562" velocity="2.0"/>
  </joint>

<!--joint M5-->
  <joint name="asibot__M5" type="revolute">
    <parent link="asibot__link4"/>
    <child link="asibot__link5"/>
    <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣0␣␣␣␣␣0.04"/>
    <axis xyz="0␣␣0␣␣1"/>
    <limit effort="-1.0" lower="-3.1416" upper="3.1416" velocity="2.0"/>
  </joint>

<!--joint M6-->
  <joint name="asibot__M6" type="fixed">
    <parent link="asibot__link5"/>
    <child link="asibot__link6"/>
    <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣0␣␣␣␣␣0"/>
    <!--<axis xyz="0␣␣0␣␣1"/>
    <limit effort="-1.0" lower="-3.1416" upper="3.1416" velocity="2.0"/>-->
  </joint>


<!--link0-->
  <link name="asibot__link0">
    <inertial>
      <mass value="1"/>
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣0␣␣0"/>
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
    <collision name="asibot__collision">
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣0␣␣0"/>
      <geometry>
        <mesh filename="file:///home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam01s.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </collision>
    <collision name="asibot__collision2">
      <origin rpy="3.14159␣␣0␣␣␣␣␣␣␣0" xyz="0␣␣␣␣␣0␣␣␣␣␣0.015"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/macho.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </collision>
    <visual name="asibot__visual">
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣0␣␣0"/>
      <geometry>
        <mesh filename="file:///home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam01s.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </visual>
    <visual name="asibot__visual2">
      <origin rpy="3.14159␣␣0␣␣␣␣␣␣␣0" xyz="0␣␣␣␣␣0␣␣␣␣␣0.015"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/macho.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </visual>
  </link>

<!--link1-->
  <link name="asibot__link1">
    <inertial>
      <mass value="1"/>
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣0␣␣0"/>
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
    <collision name="asibot__collision">
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣0␣␣␣␣␣0.012"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam02.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </collision>
    <visual name="asibot__visual">
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣0␣␣␣␣␣0.012"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam02.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </visual>
  </link>

<!--link2-->
  <link name="asibot__link2">
    <inertial>
      <mass value="1"/>
      <origin rpy="0␣␣0␣␣0.165" xyz="0␣␣0␣␣0"/>
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
    <collision name="asibot__collision">
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣␣0.0145␣-0.06"/>
```

```
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam03.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </collision>
    <visual name="asibot__visual">
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣␣0.0145␣-0.06"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam03.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </visual>
  </link>

<!--link3-->
  <link name="asibot__link3">
    <inertial>
      <mass value="1"/>
      <origin rpy="0␣␣0␣␣0.56299" xyz="0␣␣␣0␣␣0"/>
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
    <collision name="asibot__collision">
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣␣0.0145␣-0.06"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam04.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </collision>
    <visual name="asibot__visual">
      <origin rpy="0␣␣0␣␣0" xyz="0␣␣␣␣␣␣0.0145␣-0.06"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam04.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </visual>
  </link>

<!--link4-->
  <link name="asibot__link4">
    <inertial>
      <mass value="1"/>
      <origin rpy="0␣␣0␣␣0.962" xyz="0␣␣0␣␣0"/>
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
    <collision name="asibot__collision">
      <origin rpy="3.14159␣␣0␣␣␣␣␣␣0" xyz="-0.0012␣␣0.001␣␣␣0.14"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam02.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </collision>
    <visual name="asibot__visual">
      <origin rpy="3.14159␣␣0␣␣␣␣␣␣0" xyz="-0.0012␣␣0.001␣␣␣0.14"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam02.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </visual>
  </link>

<!--link5-->
  <link name="asibot__link5">
    <inertial>
      <mass value="0.33"/>
      <origin rpy="0␣␣0␣␣1" xyz="0␣␣0␣␣0"/>
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
    <collision name="asibot__collision_2">
      <origin rpy="3.14159␣␣0␣␣␣␣␣␣0" xyz="-0.0012␣␣0␣␣␣␣0.115"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam01s.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </collision>
    <collision name="asibot__collision">
      <origin rpy="0␣␣0␣␣0" xyz="-0.0012␣␣0␣␣␣␣␣␣0.097"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/macho.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </collision>
    <visual name="asibot__ModelPreview_4__link5__visual_5">
      <origin rpy="0␣␣0␣␣0" xyz="-0.0012␣␣0␣␣␣␣␣␣0.097"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/macho.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </visual>
    <visual name="asibot__visual">
      <origin rpy="3.14159␣␣0␣␣␣␣␣␣0" xyz="-0.0012␣␣0␣␣␣␣0.115"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/ensam01s.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </visual>
  </link>

<!--link6-->
  <link name="asibot__link6">
    <inertial>
```

```
      <mass value="0.33"/>
      <origin rpy="0␣␣0␣␣1" xyz="0␣␣0␣␣0"/>
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
    <collision name="asibot__collision_3">
      <origin rpy="-1.5708␣␣␣␣␣0␣␣␣␣␣0" xyz="0␣␣-0.06␣␣␣␣␣0.175"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/pinza.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </collision>
    <visual name="asibot__ModelPreview_4__link6__visual_4">
      <origin rpy="-1.5708␣␣␣␣␣0␣␣␣␣␣0" xyz="0␣␣-0.06␣␣␣␣␣0.175"/>
      <geometry>
        <mesh filename="file://home/home/celia/Documentos/tfg/Archivos_stl/stl_parque/pinza.stl" scale="0.001␣0.001␣0.001"/>
      </geometry>
    </visual>
  </link>
</robot>
```

Bibliografia

[1] Codrin Pasca. *History of robotics*. University of ottawa, Enrichment mini-course, May 5, 2003. Se puede encontrar en: https://pdfs.semanticscholar.org/4f5f/1f60b8484afc46a380cc75e3f688c6354940.pdf.

[2] Noticias Uc3m. *Desarrollan un terapeuta robótico para niños*. Abril 20, 20015. Se puede encontrar en: http://portal.uc3m.es/portal/page/portal/repositorio_noticias/noticias_generales/\Desarrollan%20un%20terapeuta%20rob%F3tico%20para%20ni%F1os?_template=/\SHARED/pl_noticias_detalle_pub.

[3] Portal Uc3m. *Maggie: futuro, autonomía y diversión*. Se puede encontrar en: http://portal.uc3m.es/portal/page/portal/actualidad_cientifica/actualidad/\reportajes/archivo_reportajes/Maggie_futuro_autonomia_diversion.

[4] F. Alonso-Martn, Arnaud A. Ramey, Miguel A. Salichs. *Maggie: el robot traductor*. Se puede encontrar en: https://www.researchgate.net/profile/Arnaud_Ramey/publication/281120796_Maggie_el_robot_traductor/links/55d78ccf08aec156b9aa166c.pdf.

[5] *Página oficial de A.M.O.R.*. Se puede encontrar en: http://www.amorrobot.com/.

[6] *Robotics lab*. Se puede encontrar en: http://www.amorrobot.com/.

[7] Alberto Jardón, Antonio Giménez,Raúl Correal,Santiago Martinez,Carlos Balaguer. *ASIBOT: robot portátil de asistencia a discapacitados. concepto, arquitectura de control y evaluación clínica*. Revista Iberoamericana de Automática e Informática, Vol 5, No 4 (2004). Se puede encontrar en: https://journals.tdl.org/jodi/index.php/jodi/article/view/149/147.

[8] *RoboticsLab — RoboHealth*. Se puede encontrar en: http://roboticslab.uc3m.es/roboticslab/project/robohealth.

[9] Alberto Jardón, Antonio Giménez,Raúl Correal,R. Cabas,Carlos Balaguer. *A Portable Light-weight ClimbingRobot for Personal Assistance Applications*. Se puede encontrar en: https://link.springer.com/chapter/10.1007/3-540-26415-9_115.

[10] Alberto Jardón, Juan G.Victores, Santiago Morante y Carlos Balanguer. *Creación de Tareas de Asistencia Robótica Mediante la Interacción Multimodal*. Actas del VII Congreso Iberoamericano de Tecnologías de Apoyo a la Discapacidad, 2013. Se puede encontrar en:http://roboticslab.uc3m.es/roboticslab/sites/default/files/Victores%20et%20al.%20-%202013%20-%20Creaci%C3%B3n%20de%20Tareas%20de%20Asistencia%20Rob%C3%B3tica%20Mediante%20la%20Interacci%C3%B3n%20Multimodal.pdf.

[11] Alberto Jardón, Concepción A Monje, Ángel M. Gil, Ana I. de la Peña y Carlos Balanguer. *Usability assessment of ASIBOT: A portable robot to aid patients with spinal cord injury. Disability and rehabilitation. Assistive technology*, Octubre (2010). Se puede encontrar en: http://www.tandfonline.com/doi/full/10.3109/17483107.2010.528144.

[12] *Web oficial de ARGos*. Se puede encontrar en: http://www.argos-sim.info/.

[13] *Web oficial de Coppelia Robotics*. Se puede encontrar en: http://www.coppeliarobotics.com/.

[14] *Web oficial de Webots*. Se puede encontrar en: https://www.cyberbotics.com/features.

[15] J. Bodner H. Wykypiel G. Wetscher T. Schmid. *First experiences with the da Vinci operating robot in thoracic surgery .European Journal of Cardio-Thoracic Surgery, Volume 25, Issue 5, 1 May 2004, Pages 844851*. Se puede encontrar en:https://academic.oup.com/ejcts/article/25/5/844/459243/First-experiences-with-the-da-VinciT-operating.

[16] Mahesh S. Raisinghani, Ally Benoit, Jianchun Ding, Maria Gomez, Kanak Gupta, Victor Gusila, Daniel Power y Oliver Schmedding. *Ambient Intelligence: Changing Forms of Human-Computer Interaction and their Social Implications*. Journal of Digital Information, Vol 5, No 4 (2004). Se puede encontrar en: https://journals.tdl.org/jodi/index.php/jodi/article/view/149/147.

[17] *Página oficila de ROS*. Se puede encontrar en: http://www.ros.org/.

[18] Enrique Fernández, Luis Sánchez Crespo, Anil Mahtani y Aaron Martinez. *Learning ROS for robotics programming*. Se puede encontrar en:https://books.google.es/books?hl=es&lr=&id=n11lCgAAQBAJ&oi=fnd&pg=PP1&dq=Learning+ROS+for+robotics+programming&ots=zXqYA2hVvN&sig=D33RIbPZsMsn0li3_4UIaMyJrg4#v=onepage&q=Learning%20ROS%20for%20robotics%20programming&f=false.

[19] *Definición de OpenRave*. Se puede encontrar en: https://sourceforge.net/projects/openrave/.

[20] *OpenRave Documentation*. Se puede encontrar en: http://openrave.org/docs/latest_stable/overview/.

[21] *Definición de Gazebo*. Se puede encontrar en: http://gazebosim.org/tutorials?tut=guided_b1&cat=.

[22] *Gazebo — Robot simulation made easy*. Se puede encontrar en: http://gazebosim.org/.

[23] *Gazebo — Make a model*. Se puede encontrar en: http://www.gazebosim.org/tutorials?tut=build_model&cat=build_robot.

[24] *Extensible Markup Language (XML)*. Se puede encontrar en: https://www.w3.org/XML/.

[25] *XML in 10 points*. Se puede encontrar en: https://www.w3.org/XML/1999/XML-in-10-points-19990327.

[26] *OpenRave Custom XML Format*. Se puede encontrar en: http://openrave.programmingvision.com/wiki/index.php/Format:XML.

[27] *OpenRave Documentation — COLLADA Robot Extensions*. Se puede encontrar en: http://openrave.org/docs/latest_stable/collada_robot_extensions/.

[28] *OpenRave Documentation — Robots Overview*. Se puede encontrar en: http://openrave.org/docs/latest_stable/robots_overview/.

[29] *Gazebo — URDF in Gazebo*. Se puede encontrar en: http://gazebosim.org/tutorials/?tut=ros_urdf.

[30] *Bitbucket — sdformat*. Se puede encontrar en: https://bitbucket.org/osrf/sdformat.

[31] *SDFormat — Specification*. Se puede encontrar en: http://sdformat.org/spec?elem=sdf&ver=1.6.

[32] *ROS Wiki — Xacro*. Se puede encontrar en: http://wiki.ros.org/xacro.

[33] *ROS Wiki — Using Xacro to Clean Up a URDF File*. Se puede encontrar en: http://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File.

[34] *Página oficial de Open Inventor*. Se puede encontrar en: http://www.openinventor.com/.

[35] *Open Inventor File Format* . Se puede encontrar en: http://web.mit.edu/ivlib/www/iv/files.html.

[36] *Información sobre archivos .iv*. Se puede encontrar en: http://oss.sgi.com/projects/inventor/.

[37] *web 3D consortium — X3D & VRML, The Most Widely Used 3D Formats*. Se puede encontrar en: http://www.web3d.org/x3d-vrml-most-widely-used-3d-formats.

[38] *Página oficial de 3D System, información sobre archivos .stl.* Se puede encontrar en: https://es.3dsystems.com/quickparts/learning-center/what-is-stl-file.

[39] *COLLADA FAQ.* Se puede encontrar en: https://www.khronos.org/collada/wiki/COLLADA_FAQ.

[40] *KHRONOS group — COLLADA Overview.* Se puede encontrar en: https://www.khronos.org/collada/.

[41] *Gazebo — Make a Mobile Robot.* Se puede encontrar en: http://gazebosim.org/tutorials?tut=build_robot.

[42] *Okino computer graphics — Polytrans.* Se puede encontrar en: https://www.okino.com/conv/conv.htm.

[43] *Fill holes in an STL file.* Se puede encontrar en: https://www.youtube.com/watch?v=Y5rjUwRC-Zo.

[44] *Repair hole in model.* Se puede encontrar en: //www.youtube.com/watch?v=qawF-_4qbxI.

[45] *Fixing impossible STL's with Meshmixer 3.1.118 BETA.* Se puede encontrar en: https://www.youtube.com/watch?v=8ECNP3WN_-Q.

[46] *Página de archivos de CAD.* Se puede encontrar en: https://grabcad.com/library/bathroom-29.

[47] *Información sobre el paquete pysdf de ROS.* Se puede encontrar en: http://wiki.ros.org/pysdf.

[48] *Blog comandos de Linux.* Se puede encontrar en: https://openwebinars.net/blog/10-comandos-basicos-para-la-terminal-de-linux/.

[49] *Página de github donde descargar el paquete "pysdf".* Se puede encontrar en: https://github.com/andreasBihlmaier/pysdf.

[50] *Explicación del comando "catkin_make".* Se puede encontrar en: http://wiki.ros.org/catkin/commands/catkin_make.

[51] *Explicación del comando "rosrun".* Se puede encontrar en: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes.

[52] *Explicación para la transformación de sdf a urdf: [ROS Q&A] How to convert from SDF to URDF robot models.* Se puede encontrar en: https://www.youtube.com/watch?v=8g5nMxhi_Pw.

[53] *Bug: Mesh file path in URDF. Cannot locate resource file://.* Se puede encontrar en: https://answers.ros.org/question/39116/bug-mesh-file-path-in-urdf-cannot-locate-resource-file/.

[54] *Comando "check_urdf".* Se puede encontrar en: https://answers.ros.org/question/9208/how-to-check-xacro-syntax-like-check_urdf/ o en: http://manpages.ubuntu.com/manpages/trusty/man1/check_urdf.1.html.

[55] *Instalación de MoveIt.* Se puede encontrar en: http://moveit.ros.org/install/.

[56] *MoveIt — Concepts.* Se puede encontrar en: http://moveit.ros.org/documentation/concepts/.

[57] *Orocos Kinematics and Dynamics — KDL Documentation.* Se puede encontrar en: http://www.orocos.org/kdl.

[58] *Asistente de MoveIt.* Se puede encontrar en: http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html.

[59] *Definición de Rviz.* Se puede encontrar en: http://sdk.rethinkrobotics.com/wiki/Rviz.

[60] *Explicación del comando "roslaunch".* Se puede encontrar en: http://gazebosim.org/tutorials?tut=ros_roslaunch.

[61] *MoveIt! RViz Plugin Tutorial.* Se puede encontrar en: http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/ros_visualization/visualization_tutorial.html.

[62] Alberto Jardón Huete. *Metodología de diseo de robots de asistenciales. Aplicación al robot portátil ASIBOT.* Se puede encontrar en: https://e-archivo.uc3m.es/handle/10016/787.

[63] *GitHub — pr2_simulator.* Se puede encontrar en: https://github.com/danepowell/pr2_simulator.

[64] Lentin Joseph. *Mastering ROS for Robotics Programming.* Se puede encontrar en:http://proquest.safaribooksonline.com.biblioteca5.uc3m.es/9781783551798.

[65] *ROS — BuildingPackages.* Se puede encontrar en:http://wiki.ros.org/ROS/Tutorials/BuildingPackages.

[66] *GitHub — Códigos robot "seven_dof_arm".* Se puede encontrar en:https://github.com/qboticslabs/mastering_ros/tree/master/chapter_4_codes.

[67] *ROS Answers —Gazebo controller spawner warning.* Se puede encontrar en:https://answers.ros.org/question/214712/gazebo-controller-spawner-warning/.

[68] *ROS Answers — No valid hardware interface element found in joint* . Se puede encontrar en:https://answers.ros.org/question/186681/no-valid-hardware-interface-element-found-in-joint/.

[69] *ROS — Why ROS?.* Se puede encontrar en:http://www.ros.org/is-ros-for-me/.

[70] *GitHub.* Se puede encontrar en:https://github.com/roboticslab-uc3m/asibot-openrave-models.